CSC320 Notes

Last updated January 20, 2024

1 Image Transformations

Transformations we can do to images:

- Scaling
- Warping (preserves straight lines)
  - The basic transformation that is used to scan documents; identify the corners (perhaps by hand) which should be enough to do the warp
  - A homography / linear transformation

What are the class of transformations used to perform the operation? First, we need to know more about affine transforms.

Summary

- Two homogeneous coordinates are the same if they're multiples of each other (except 0)
- A point at infinity is where the last element of a homogenous coordinate is 0; can be represented as an angle from 0 to 180 degrees
- We can represent a line by a vector $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$ such that $\begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = 0$ or more familiarly $ax + by + c = 0$
- Given points in homogeneous coordinates p_1, p_2, the homogeneous coordinates of the line that passes through them is $p_1 \times p_2$
- Given two lines in homogeneous coordinates, their point of intersection is $l_1 \times l_2$
- Convert a homogeneous point coordinate to a regular coordinate by scaling it so that the last element is 1, then remove the last element
- Affine transformations preserve parallelism, and look like $\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & g \end{bmatrix}$

1.1 Notation Conventions

- Assume that an image is continuous. This means accessing a point on the image can be done with continuous $\mathbb{R}$ values.
- **Points are represented using column vectors:** $\begin{bmatrix} x \\ y \end{bmatrix}$, bottom 0 top image height
  - Row vectors are matrices that only contain a single row: $\begin{bmatrix} x & y \end{bmatrix}$
  - Transposing: $\begin{bmatrix} x \\ y \end{bmatrix}^T = \begin{bmatrix} x & y \end{bmatrix}$
- Homogenous coordinate representation of any point $p \in \mathbb{R}^2$: Euclidean coordinate to homogeneous coordinates
  - $\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$
  - When we represent a point in homogeneous coordinates, we don't represent that point with that vector only. It's this vector and any scaled version of this vector, all represent the same 2D point. In other words, for any $\lambda \in \mathbb{R} \setminus \{0\}$, $\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ represents the same 2D point.
  - $p \cong \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \cong \begin{bmatrix} -2x \\ -2y \\ -2 \end{bmatrix} \cong \begin{bmatrix} 2x \\ 2y \\ 2 \end{bmatrix}$
  - Two vectors of homogeneous coordinates are called equal if they represent the same 2D point.
  - $\begin{bmatrix} x \\ y \\ w \end{bmatrix} \cong \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} \Leftrightarrow \exists \lambda \neq 0, \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \lambda \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix}$
- Homogeneous coordinates to Euclidean coordinates:

$$- \frac{1}{c} \begin{bmatrix} a \\ b \\ c \end{bmatrix} [0:2]$$

1.2 Points at Infinity

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \to \begin{bmatrix} \infty \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \to \begin{bmatrix} \infty \\ \infty \end{bmatrix}$$

With homogeneous coordinates, we have a finite representation of a point that is infinitely far away. We can represent a point infinitely away only using $\mathbb{R}$. This leads to very stable geometric computations.

Points at infinity have their last coordinate equal to 0. Points at infinity are also called *ideal points* in textbooks.

What do points at infinity represent? The space described by these homogenous coordinates are called a projected plane: the Euclidean plane and a bit more.

You can encode them as a clock position with arrows on both hands (0-180 degrees)

1.3 Homogeneous 2D Line Coordinates

How do we represent a line?

We have a line l on the plane. Suppose there is a point p that lies on the line: $p \cong \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$.

What's the most general equation for a line?

$$ax + by + c = 0$$

($y = mx + b$ cannot encode vertical lines) In matrix form, the homogeneous coordinates of a line can be represented by:

$$\begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$

So, a line can also be represented by this. If you multiply this equation by any non-zero scalar, the line remains the same. Meaning for all $\lambda \neq 0$, this represents the same line.

$$\lambda \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$

The vector $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$ is the vector holding the line coordinate. It can be interpreted in any way.

1.3.1 Line Coordinates Conversion Examples

What are the homogeneous coordinates of the line $y = x$? They're written in $l^T p = 0$

$$\begin{aligned} & y = x \\ & \iff -x + y = 0 \\ & \iff -x + y + 0(1) = 0 \\ & \iff \begin{bmatrix} -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0 \end{aligned}$$

$\begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$ are the homogeneous coordinates of this line.

1.3.2 Coordinates of the line passing through two points

What are the homogeneous coordinates of the line that connects two points?

The setup, given p_1, p_2:

$$l^T p = 0$$

The line passes through two points, so they must satisfy the line equation: l must satisfy $l^T p_1 = 0$, $l^T p_2 = 0$

The fact that $l^T p_1 = 0$, $l^T p_2 = 0$ implies that l must be perpendicular to p_1 and p_2, so it means that l must be the cross product between the two

So, the general expression is, if I know the homogeneous coordinates of p_1 and p_2, then I can get the homogeneous coordinates of the line that passes through both.

$$l = p_1 \times p_2$$

So, given two image points $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$?

1. Convert to homogeneous coordinates
2. Compute the cross product

This gives us an immediate expression for the equation of the line.

1.3.3 Calculating the cross product

As a matrix-vector product:

$$p_1 \times p_2 = \begin{bmatrix} 0 & -z_1 & y_1 \\ z_1 & 0 & -x_1 \\ -y_1 & x_1 & 0 \end{bmatrix} p_2$$

So, we have an analytical expression for computing the line coordinates from two points.

Alternatively, as a determinant:

$$p_1 \times p_2 = \begin{vmatrix} i & j & k \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix}$$
$$= i \begin{vmatrix} y_1 & z_1 \\ y_2 & z_2 \end{vmatrix} - j \begin{vmatrix} x_1 & z_1 \\ x_2 & z_2 \end{vmatrix} + k \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix}$$

i, j, k are short hands for vectors. $i = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$, $j = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$, $k = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$.

Feel free to use whatever you want for cross products.

1.4 Coordinates of the Intersection of Two Lines

We have two lines that we know, and we want to find the homogeneous coordinates of their intersection.

We know line $l_1 = \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix}$, $l_2 = \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix}$. Find $p = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$. It must satisfy:

$$l_1^T p = 0,\ l_2^T p = 0$$

So, what is p? It's the cross product.

$$p = l_1 \times l_2$$

We have a very easy way to compute intersections.

1.4.1 In case they're parallel

Now, what happens when the two lines are **parallel?** You'll get a point at infinity, with a 0 at the third coordinate. Here's an example given $y = 1$, $y = 2$:

$$l_1 = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix}, \; l_2 = \begin{bmatrix} 0 \\ 1 \\ -2 \end{bmatrix}$$

We have the homogeneous coordinates of two lines. Their intersection is going to be the cross product, $l_1 \times l_2$. What's the result?

$$l_1 \times l_2 = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ -2 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}$$

This represents negative infinity at the x-axis. Everything is completely finite from the perspective of homogeneous coordinates, but this is a point of infinity. In a different direction, you would get a different point at infinity. This is how we can check if two lines are parallel. If their intersection computed through the cross product gives us 0 at the last coordinate, they have to be parallel.

The order of the terms in the cross product do not matter due to how homogeneous coordinates work. A point at infinity could be seen as a clock with a handle that points in both directions

1.5 Affine Transformations

A matrix can transform all vectors in a space.

Scaling: Where x is the scale of x

$$\begin{bmatrix} x & 0 \\ 0 & y \end{bmatrix}$$

Shearing:

$$\begin{bmatrix} 1 & \text{horizontal shear} \\ \text{vertical shear} & 1 \end{bmatrix}$$

Rotations: shear horizontally and vertically by the same amount. The cosine function prevents size changes from rotating.

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Translation: requires homogeneous coordinates

$$\begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Chaining multiplications **compose transformations**, the order being from right to left. You can encode many transformations in a single matrix. However, regardless of how you multiply, as long as you are multiplying transform matrices, it they will **always** be in this form:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

The last row will **always be** $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$. It may look like $\begin{bmatrix} 0 & 0 & g \end{bmatrix}$ in some cases.

We can multiply this entire matrix by any scalar except 0 and the homogeneous transformation would remain the same.

Most general affine transform matrix:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & g \end{bmatrix}$$

1.5.1 Geometric Properties Preserved by Affine Transformations

PRESERVED

- Parallelism
  - Parallel line pairs stay parallel to each other. Remember, parallel lines intersect at infinity
  - The transformation maps point at infinity to points at infinity (it may not be the same point at infinity)
  - Why? Because check the bottom row $\begin{bmatrix} 0 & 0 & g \end{bmatrix}$, the first two are 0
  - What if they were not? Then, some non-infinity points might be mapped to infinity and vice versa. This implies that parallelism would not be preserved (possibly but not always a 3D rotation could do this).

NOT PRESERVED

- Angles
- Lengths

1.6 Projective Transforms

Any 2D transform of homogeneous coordinates that is **represented by an invertible** 3×3 **matrix**. Known as Homography. It will **not** preserve parallelism.

For example, the way scanner apps distort images is a projective transform.

This is a fundamental distinction between general linear transformations from affine transformations. Affine transformations are much more restrictive; scanner apps can't use affine transforms by themselves.

The scanning procedure depends on figuring out what is the homography the 3×3 matrix that allows us to take a raw image and convert it to a proper, scanned image.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ l & m & g \end{bmatrix}$$

Homographies preserve linearity. Any lines that lie before a homography, will remain on another line after a homography.

1.7 Forward Mapping Algorithm

Suppose that we have the homography transformation matrix H. Now, create an algorithm that creates an image given an old image.

There is a bunch of ways to do it. There's a very easy way that doesn't give great results, and there's a slight tweak that solves it.

Our input: `src_image`, H

Output: `dest_image`.

How does this work? Let's see how the array of pixels can be represented as points on a 2D plane. Every pixel is just a (row, column) coordinate and we need to convert it to an (x, y) point on the plane.

So, the forward mapping algorithm:

```
for c=1 to num_columns
  for r=1 to num_rows:
    x, y = pixel_xy(r, c) # get the source pixel's (x, y)
       coordinates
    p = homogeneous_coords(x, y)
    p_prime = H * p
    x_prime, y_prime = euclidean_coords(p_prime)
    r_prime, c_prime = pixel_rc(x_prime, y_prime) # floor,
       round, ceiling, I don't care for now
    dest_image(r_prime, c_prime) = src_image(r, c)
```

This algorithm already has problems.

If I stretch the image, we could have gaps that will never be filled. A lot of my pixels in the destination image will contain nothing.

If I shrink the image, I can have the opposite problem. Two pixels from the source might map overwrite an existing written pixel in the destination image. We're losing information; not a good thing either.

It's possible to have both happen in the same image.

There's a very simple fix for this: an algorithm that doesn't go forward; it goes backwards.

1.8 Backward Mapping Algorithm

We have a loop that goes over the destination pixels. We go over the (x, y) coordinates in the destination image and use the inverse homography H^{-1} to figure out what pixel to target in the source image.

Because we are looping over the destination pixels, we can look at every single pixel in the destination image, and we'll get a value for every one of them. The destination image will get filled. There will be no gaps, regardless of whether we're doing magnification, stretching, and so on.

This means that will every single pixel in the destination image be filled. **No**, there could be blank pixels. A pixel in the destination image might map to a pixel **outside** the source image.

```
for c_prime=1 to num_columns
  for r_prime=1 to num_rows:
     x_prime, y_prime = pixel_xy(r_prime, c_prime)
     p_prime = homogeneous(x_prime, y_prime)
     p = inverse(H)*p_prime
     x, y = euclid(p)
     r, c = pixel_rc(x, y)
     des_image(r_prime, c_prime) = source[r, c]
```

2 Image Projection

How do we relate 3D points in the world to 2D pixels in an image?

We'll look at

- The geometry of perspective projection and concepts of center of projection, focal length
- How to represent 3D rays and points in homogeneous 3D coordinates
- Proving that all perspective images of a plane can be stitched via Homographies
- Why is it that homography warping is enough
- How do we estimate the Homographies we need?
- Understanding what 3D information is lost by perspective projection
- Making 3D measurements on a planar surface by warping its photo to a canonical view
- So on

2.1 Camera Aperture

Why do cameras have an aperture? Why not just have the sensor and nothing in part of it? Isn't that good enough?

Can't orthographically ray-trace. You'll get an extremely blurry image. All pixels receive light from all possible points.

So, instead, let's use an aperture. This results in a 1-1 correspondence between world points and image points, ideally. Like what you've been told, you'll get an upside down horizontally reflected image. This way of getting images has been known for hundreds of years; this is called a camera obscura.

2.1.1 Adjusting the Aperture

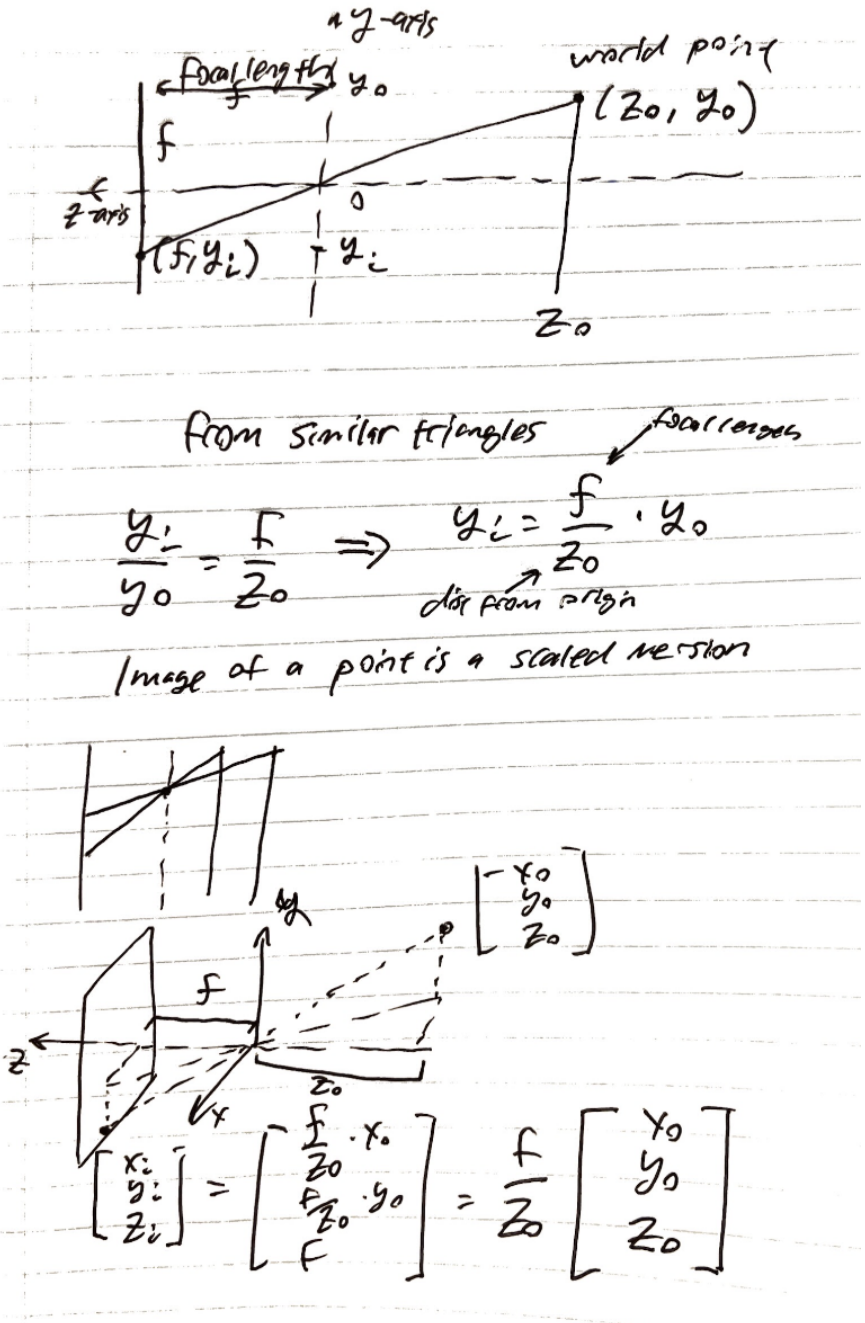
- ↑ Aperture size, ↑ Blurriness

2.2 Geometry of Perspective Projection

The focal length is the distance between the aperture and the image plane. Decreasing the focal length gives us a smaller image. It's a number that describes magnification.

- ↑ Focal length, ↑ image size, ↓ field of view

The lower the focal length, the image is being concentrated in a smaller set of pixels if the sensor's pixel count remains constant. Yet, it also gives us more field of view.



$$\begin{bmatrix} x_i \\ y_i \\ z_i = f \end{bmatrix} = \frac{f}{z_0} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

Where:

- $\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}$ is the location on the camera sensor
- $\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$ is the location of the actual image
- f is the focal length
- For the purposes of axis-alignment, $-z_0$ is the distance from the aperture to the image.
- $\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ is the pinhole / apeature

Observation: a lower magnitude of z_0 (but constant x_0, y_0) means the object is closer to the camera, so it appears larger on the sensor (points are more spread out)

If we scale $\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$ by a constant factor, how the image is projected **will look the same.**

This is how we can think of 2D homogeneous coordinates (think of superliminal).
Interpretation of homogeneous equality: All 3D points having the same projection:

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \cong \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

2.3 Representing 3D Points in Homogeneous Coordinates

3D coordinates with scale invariance can only represent rays. So, how do we represent 3D points? Introducing homogeneous 3D coordinates, like always, defined up to a scale factor.

$$\underbrace{\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}}_{\text{euclidean}} \rightarrow \underbrace{\begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ w_0 \end{bmatrix}}_{\text{homogeneous}} \rightarrow \underbrace{\frac{1}{w_0} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}}_{\text{euclidean}}$$

And guess what? $\begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 0 \end{bmatrix}$ represents a point at infinity. Homogeneous coordinates allow us to represent points at infinity in 3D.

Let's look at how homogeneous coordinates help us simplify the expression for perspective projection.

Let's expand our *point in world space to sensor space* transformation using our new homogeneous coordinates system:

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{f_0}{z_0} x_0 \\ \frac{f_0}{z_0} y_0 \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix}$$

(We can change the scaling of the input $\begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix}$ and we would still get the "same" sensor space coordinate.)

2.4 Alignment and Stitching

Let's talk about images of specific geometric objects (like lines and planes). A very important property of prospective project is that it preserves linearity. All the lines

that were straight in the original source is straight in the output. It falls from the fact that our transformation has a matrix in between – it's a linear transformation. But it's also possible to reason geometrically why lines in the world map lines to the image.

In order to take two photographs of the same object in two different viewpoints and stitch them together (in order to do a homography), **two conditions must hold for the homography:**

- Lines map to lines
- Each line in one image is transformed to a **unique** line in the other (invertibility) – in other words, we can reverse the transformation (minus loss of quality).

2.4.1 Linearity of Perspective Projection

Why does perspective projection preserve linearity?

Projection goes through a center of projection. Every point of a line gets mapped to the sensor along the center of projection.

2.5 When can I Stitch Together?

- If the image can be aligned
- Viewpoint must remain the same (same lines **in real life**) must map to the same place in the sensor – **preserve the center of projection.** You may rotate your camera, but your camera **must be anchored at the center of projection (usually the pinhole).**
  - Beware, an iPhone panorama is **not** that because you are moving your phone very far. It's not a real photograph anymore. You wouldn't be able to create a camera with a wider field of view and capture the same image.
  - That blue fence in the slide is curved not because the requirements for stitching failed – it's some post processing just for visual intent. Maybe your sensor is not planar – that doesn't matter.

  - Your Minecraft screenshots by moving your character's camera angle can be stitched together (with some assumptions I'm making).
  - **What a 360 camera can do**

- Photos taken from pure camera rotation can be aligned and stitched

Place the camera (or at least the center of projection) in the same place and you can stitch.

Radial distortions break linearity. Images that have non-linear distortions are not stitch-able without first undistorting (correcting) them. This could be an artifact of the actual lens. The image plane (sensor) **is** a flat surface, but because you're trying to get an image with a very wide FOV, you must pack all that information on a flat surface, and it is the lens that creates these distortions.

Because these distortions do not depend on what the lens is taking a picture of, they can be undone. This is called radial distortion correction.

2.6 How do we compute Homographies?

Let's see.

The big picture – as long as you can find 4 points in one image, you can compute H – a 3×3 matrix that maps one image to the other.

A single correspondence means you have some point out there and you can map it somewhere else.

$$\underbrace{\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix}}_{\text{known}} \cong \underbrace{\begin{bmatrix} a & b & c \\ d & e & f \\ h & k & 1 \end{bmatrix}}_{\text{unknown : } H} \begin{bmatrix} x_i \\ y_i \\ \underbrace{1}_{\text{known}} \end{bmatrix}$$

Make some conversions:

$$x_i' (hx_i + ky_i + 1) = ax_i + by_i + c$$
$$y_i' (hx_i + ky_i + 1) = dx_i + ey_i + f$$

If 1 point correspondence gives us two equations, a 4-point correspondence gives us **8 equations and 8 unknowns.** That gives us our homography. That is our source and destination.