PART FIVE

GEOMETRY WITH ONE CAMERA

CHAPTER 11

Camera Matrices

11.1 SIMPLE PROJECTIVE GEOMETRY

Draw a pattern on a plane, then view that pattern with a perspective camera. The distortions you observe are more interesting than are predicted by simple rotation, translation and scaling. For example, if you drew parallel lines, you might see lines that intersect at a vanishing point – this doesn't happen under rotation, translation and scaling. *Projective geometry* can be used to describe the set of transformations produced by a perspective camera.

11.1.1 Homogeneous Coordinates and Projective Spaces

The coordinates that every reader will be most familiar with are known as *affine coordinates*. In affine coordinates, a point on the plane is represented by 2 numbers, a point in 3D is represented with 3 numbers, and a point in k dimensions is represented with k numbers. Now adopt the convention that a point in k dimensions is represented by $k+1$ numbers *not all of which are zero*. Two representations $\mathbf{X}_1$ and $\mathbf{X}_2$ represent the same point (write $\mathbf{X}_1 \equiv \mathbf{X}_2$) if there is some $\lambda \neq 0$ so that

$$\mathbf{X}_1 = \lambda \mathbf{X}_2.$$

These coordinates are known as *homogeneous coordinates*, and will offer a particularly convenient representation of perspective projection.

Remember this: *In homogeneous coordinates, a point in a k dimensional space is represented by $k+1$ coordinates $(X_1, \ldots, X_{k+1})$, together with the convention that*

$$(X_1, \ldots, X_{k+1}) \equiv \lambda(X_1, \ldots, X_{k+1}) \text{ for } \lambda \neq 0.$$

The space represented by $k+1$ homogeneous coordinates is different from the space represented by k affine coordinates in important but subtle ways. We start with a 1D space. In homogenous coordinates, we represent a point on a 1D space with two coordinates, so (X_1, X_2) (by convention, homogeneous coordinates are written with capital letters). Two sets of homogeneous coordinates (U_1, U_2) and (V_1, V_2) represent different points if there is no $\lambda \neq 0$ such that $\lambda(U_1, U_2) = (V_1, V_2)$. Now consider the set of all the distinct points, which is known as the *projective line*. Any point on an ordinary line (the *affine line*) has a corresponding point on the projective line. In affine coordinates, a point on the affine line is given by a single coordinate x. This point can be identified with the point on the projective line

given by $(X_1, X_2) = \lambda(x, 1)$ (for $\lambda \neq 0$) in homogeneous coordinates. Notice that the projective line has an "extra point" – $(X_1, 0)$ are the homogeneous coordinates of a single point (check this), but this point would be "at infinity" on the affine line.

Example: 11.1 *Seeing the point at infinity*

You can actually see the point at infinity. Recall that lines that are parallel in the world can intersect in the image at a vanishing point. This vanishing point turns out to be the image of the point "at infinity" on the parallel lines. For example, on the plane $y = -1$ in the camera coordinate system, draw two lines $(1, -1, t)$ and $(-1, -1, t)$ (these lines are in Figure 32.2). Now these lines project to $(f1/t, f(-1/t), f)$ and $(f(-1/t), f(-1/t), f)$ on the image plane, and their vanishing point is $(0, 0, f)$. This vanishing point occurs when the parameter t reaches infinity. The exercises work this example in homogeneous coordinates.

There isn't anything special about the point on the projective line given by $(X_1, 0)$. You can see this by identifying x on the affine line with $(X_1, X_2) = \lambda(1, x)$ (for $\lambda \neq 0$). Now $(X_1, 0)$ is a point like any other, and $(0, X_2)$ is "at infinity". A little work establishes that there is a 1-1 mapping between the projective line and a circle (exercises).

Higher dimensional spaces follow the same pattern. In affine coordinates, a point in a k dimensional affine space (eg an *affine plane*; *affine 3D space*; etc) is given by k coordinates $(x_1, x_2, \ldots, x_k)$. The space described by $k + 1$ homogeneous coordinates is a *projective space* (a *projective plane*; *projective 3D space*; etc). A point $(x_1, x_2, \ldots, x_k)$ in a k dimensional affine space can be identified with $(X_1, X_2, \ldots, X_{k+1}) = \lambda(x_1, x_2, \ldots, x_k, 1)$ (for $\lambda \neq 0$) in the k dimensional projective space. The points in the projective space given by $(X_1, X_2, \ldots, 0)$ have no corresponding points in the affine space. Notice that this set of points is a $k - 1$ dimensional space in homogeneous coordinates. When $k = 2$, this set is a projective line, and is referred to as the *line at infinity*, and the whole space is known as the *projective plane*. As the exercises show, you can see the line at infinity: the horizon of a plane in the image is actually the image of the line at infinity in that plane.

When $k = 3$, this set is itself a projective plane, and is known as the *plane at infinity*; the whole space is sometimes known as *projective 3-space*. Notice this means that 3D projective space is obtained by "sewing" a projective plane to the 3D affine space we are accustomed to. The piece of the projective space "at infinity" isn't special, using the same argument as above. The particular line (resp. plane) that is "at infinity" is chosen by the homogeneous coordinate you divide by. There is an established convention in computer vision of dividing by the last homogeneous coordinate and talking about the line at infinity and the plane at infinity.

Remember this: *The k dimensional space represented by $k+1$ homogeneous coordinates is a projective space. You can represent a point $(x_1, \ldots, x_k$ in affine k space in this projective space as $(x_1, \ldots, x_k, 1)$. Not every point in the projective space can be obtained like this – the points $(X_1, \ldots, X_k, 0)$ are "extra". These points form a projective $k-1$ space which is thought of as being "at infinity". Important cases are $k = 1$ (the projective line with a point at infinity); $k = 2$ (the projective plane with a line at infinity).*

11.1.2 Lines and Planes in Projective Space

Lines on the affine plane form one important example of homogeneous coordinates. A line is the set of points (x, y) where $ax + by + c = 0$. We can use the coordinates (a, b, c) to represent a line. If $(d, e, f) = \lambda(a, b, c)$ for $\lambda \neq 0$ (which is the same as $(d, e, f) \equiv (a, b, c)$), then (d, e, f) and (a, b, c) represent the same line. This means the coordinates we are using for lines are homogeneous coordinates, and the family of lines in the affine plane is a projective plane. Notice that encoding lines using affine coordinates must leave out some lines. For example, if we insist on using $(u, v, 1) = (a/c, b/c, 1)$ to represent lines, the corresponding equation of the line would be $ux + vy + 1 = 0$. But no such line can pass through the origin – our representation has left out every line through the origin.

Lines on the projective plane work rather like lines on the affine plane. Write the points on our line using homogeneous coordinates to get

$$(x, y, 1) = (X_1/X_3, X_2/X_3, 1)$$

or equivalently (X_1, X_2, X_3) where $X_1 = xX_3$, $X_2 = yX_3$. Substitute to find the equation of the corresponding line on the projective plane, $aX_1 + bX_2 + cX_3 = 0$, or $\mathbf{a}^T\mathbf{X} = 0$. There is an interesting point here. A set of three homogenous coordinates can be used to describe either a point on the projective plane or a line on the projective plane.

Remember this: *A line on the projective plane is the set of points $\mathbf{X}$ such that*

$$\mathbf{a}^T\mathbf{X} = 0.$$

Here $\mathbf{a}$ is a vector of homogeneous coordinates specifying the particular line.

Remember this: *Write* $\mathbf{P}_1$ *and* $\mathbf{P}_2$ *for two points on the projective plane that are represented in homogeneous coordinates and are different. From the exercises, the line through these two points is given by*

$$\mathbf{a} = \mathbf{P}_1 \times \mathbf{P}_2.$$

From the exercises, a parametrization of this line is given by

$$U\mathbf{P}_1 + V\mathbf{P}_2.$$

Planes in projective 3-space work rather like lines on the projective plane. The locus of points (x, y, z) where $ax + by + cz + d = 0$ is a plane in affine 3-space. Because (a, b, c, d) and $\lambda(a, b, c, d)$ give the same plane, we have that (a, b, c, d) are homogeneous coordinates for a plane in 3D. We can write the points on the plane using homogeneous coordinates to get

$$(x, y, z, 1) = (X_1/X_4, X_2/X_4, X_3/X_4, 1)$$

or equivalently

$$(X_1, X_2, X_3, X_4) \text{ where } X_1 = xX_4, X_2 = yX_4, X_3 = zX_4.$$

Substitute to find the equation of the corresponding plane in projective 3-space $aX_1 + bX_2 + cX_3 + dX_4 = 0$ or $\mathbf{a}^T\mathbf{X} = 0$. A set of four homogenous coordinates can be used to describe either a point in projective 3-space or a plane in projective 3-space.

Remember this: *A plane in projective 3D is the set of points* $\mathbf{X}$ *such that*

$$\mathbf{a}^T\mathbf{X} = 0.$$

Here $\mathbf{a}$ *is a vector of homogeneous coordinates specifying the particular plane.*

Remember this: *Write $\mathbf{P}_1$, $\mathbf{P}_2$ and $\mathbf{P}_3$ for three points in projective 3D that are represented in homogeneous coordinates, are different points, and are not collinear. From the exercises, the plane through these points is given by*

$$\mathbf{a} = \mathit{NullSpace}\left(\begin{bmatrix} \mathbf{P}_1^T \\ \mathbf{P}_2^T \\ \mathbf{P}_3^T \end{bmatrix}\right).$$

From the exercises, a parametrization of this plane is given by

$$U\mathbf{P}_1 + V\mathbf{P}_2 + W\mathbf{P}_3.$$

11.1.3 Homographies

Write $\mathbf{X} = (X_1, X_2, X_3)$ for the coordinates of a point on the projective plane. Now consider $\mathbf{V} = \mathcal{M}\mathbf{X}$, where $\mathcal{M}$ is a 3×3 matrix with non-zero determinant. We can interpret $\mathbf{V}$ as a point on the projective plane, and in fact $\mathcal{M}$ is a mapping from the projective plane to itself. There is something to check here. Write $\mathcal{M}(\mathbf{X})$ for the point that $\mathbf{X}$ maps to, etc. Because $\mathbf{X} \equiv \lambda\mathbf{X}$ (for $\lambda \neq 0$), we must have that $\mathcal{M}(\mathbf{X}) \equiv \mathcal{M}(\lambda\mathbf{X})$ otherwise one point would map to several points. But

$$\mathcal{M}(\mathbf{X}) = \mathcal{M}\mathbf{X} \equiv \lambda\mathcal{M}\mathbf{X} = \mathcal{M}(\lambda\mathbf{X})$$

so $\mathcal{M}$ is a mapping. Such mappings are known as *homographies*. You should check that $\mathcal{M}^{(-1)}$ is the inverse of $\mathcal{M}$, and is a homography. You should check that $\mathcal{M}$ and $\lambda\mathcal{M}$ represent the same homography. Homographies are interesting to us because any view of a plane by a perspective (or orthographic) camera is a homography, and a variety of useful tricks rest on understanding homographies.

Any homography will map every line to a line. Write $\mathbf{a}$ for the line in the projective plane whose points satisfy $\mathbf{a}^T\mathbf{X} = 0$. Now apply the homography $\mathcal{M}$ to those points to get $\mathbf{V} = \mathcal{M}\mathbf{X}$. Notice that

$$\mathbf{a}^T\mathcal{M}^{(-1)}\mathbf{V} = \mathbf{a}^T\mathbf{X} = 0,$$

so that the line $\mathbf{a}$ transforms to the line $\mathcal{M}^{(-T)}\mathbf{a}$. Homographies are easily inverted.

Remember this: *A homography is a mapping from the projective plane to the projective plane. Assume $\mathcal{M}$ is a 3×3 matrix with non-zero determinant; then the homography represented by $\mathcal{M}$ maps the point with homogeneous coordinates $\mathbf{X}$ to the point with homogeneous coordinates $\mathcal{M}\mathbf{X}$. The two matrices $\mathcal{M}$ and $\lambda\mathcal{M}$ represent the same homography, and the inverse of this homography is represented by $\mathcal{M}^{-1}$. The homography represented by $\mathcal{M}$ will map the line represented by $\mathbf{a}$ to the line represented by $\mathcal{M}^{-T}\mathbf{a}$.*

11.2 CAMERA MATRICES AND TRANSFORMATIONS

11.2.1 Perspective and Orthographic Camera Matrices

In affine coordinates we wrote perspective projection as $(X, Y, Z) \rightarrow (X/Z, Y/Z)$ (remember, we will account for f later). Now write the 3D point in homogeneous coordinates, so

$$\mathbf{X} = (X_1, X_2, X_3, X_4) \text{ where } X_1 = XX_4, \text{ etc.}$$

Write the point in the image plane in homogeneous coordinates as well, to obtain

$$\mathbf{I} = (I_1, I_2, I_3) \text{ where } I_1 = (X/Z)I_3 \text{ and } I_2 = (Y/Z)I_3.$$

So we could use

$$\mathbf{I} = (X, Y, Z) \equiv (X/Z, Y/Z, 1) \equiv (X_1/X_4, X_2/X_4, X_3/X_4) \equiv (X_1, X_2, X_3).$$

Notice that (X, Y, Z) is a natural choice of homogeneous coordinates for the point in the image plane. This means that, in homogeneous coordinates, we can represent perspective projection as

$$(X_1, X_2, X_3, X_4) \rightarrow (X_1, X_2, X_3) \equiv (X_1, X_2, X_3).$$

or

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}$$

where the matrix is known as the *perspective camera matrix* (write $\mathcal{C}_p$). Notice that this representation preserves the property that the focal point of the camera cannot be imaged, and is the only such point. The focal point can be represented in homogeneous coordinates by $(0, 0, 0, T)$, for $T \neq 0$. This maps to $(0, 0, 0)$, which is meaningless in homogeneous coordinates. You should check no other point maps to $(0, 0, 0)$.

Remember this: *The perspective camera matrix is*

$$\mathcal{C}_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

In affine coordinates, in the right coordinate system and assuming that the scale is chosen to be one, scaled orthographic perspective can be written as $(X, Y, Z) \rightarrow (X, Y)$. Following the argument above, we obtain in homogeneous coordinates

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}$$

where the matrix is known as the *orthographic camera matrix* (write $\mathcal{C}_o$).

Remember this: *The orthographic camera matrix is*

$$\mathcal{C}_o = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

11.2.2 Cameras in World Coordinates

The camera matrix describes a perspective (resp. orthographic) projection for a camera in a specific coordinate system – the focal point is at the origin, the camera is looking down the z-axis, and so on. In the more general case, the camera is placed somewhere in world coordinates looking in some direction, and we need to account for this. Furthermore, the camera matrix assumes that points in the camera are reported in a specific coordinate system. The pixel locations reported by a practical camera might not be in that coordinate system. For example, many cameras place the origin at the top left hand corner. We need to account for this effect, too.

A general perspective camera transformation can be written as:

$$\begin{aligned} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} &= \begin{bmatrix} \text{Transformation} \\ \text{mapping image} \\ \text{plane coords to} \\ \text{pixel coords} \end{bmatrix} \mathcal{C}_p \begin{bmatrix} \text{Transformation} \\ \text{mapping world} \\ \text{coords to camera} \\ \text{coords} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} \\ &= \mathcal{T}_i \mathcal{C}_p \mathcal{T}_e \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} \end{aligned}$$

The parameters of $\mathcal{T}_i$ are known as *camera intrinsic parameters* or *camera intrinsics*, because they are part of the camera, and typically cannot be changed. The parameters of $\mathcal{T}_e$ are known as *camera extrinsic parameters* or *camera extrinsics*, because they can be changed.

11.2.3 Camera Extrinsic Parameters

The transformation $\mathcal{T}_e$ represents a rigid motion (equivalently, a *Euclidean transformation*, which consists of a 3D rotation and a 3D translation). In affine coordinates, any Euclidean transformation maps the vector $\mathbf{x}$ to

$$\mathcal{R}\mathbf{x} + \mathbf{t}$$

where $\mathcal{R}$ is an appropriately chosen 3D rotation matrix (check the endnotes if you can't recall) and $\mathbf{t}$ is the translation. Any map of this form is a Euclidean

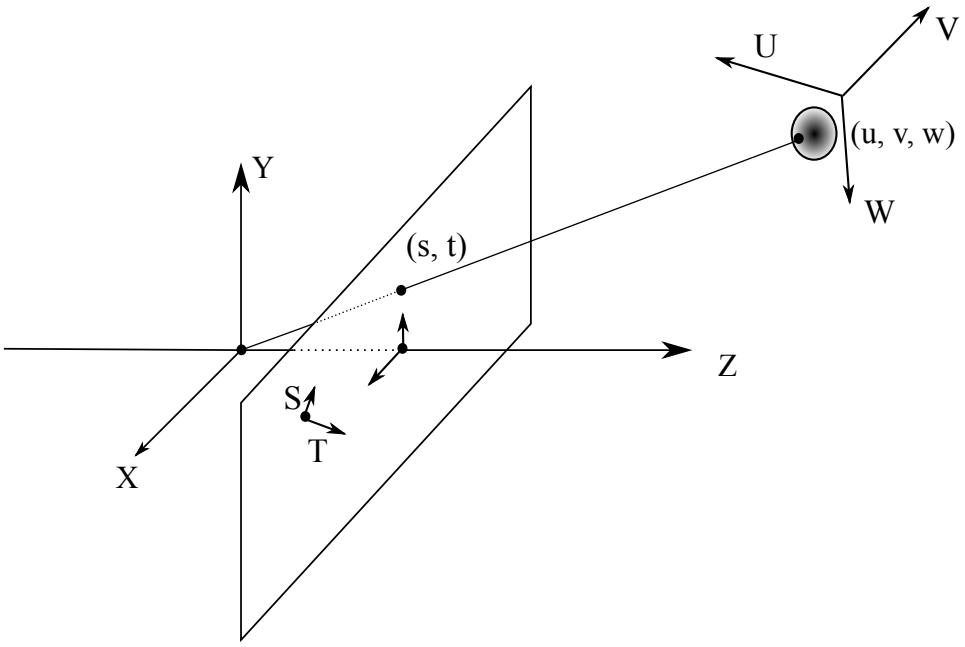

FIGURE 11.1: *A perspective camera (in its own coordinate system, given by X, Y and Z axes) views a point in world coordinates (given by (u, v, w) in the UVW coordinate system) and reports the position of points in ST coordinates. We must model the mapping from (u, v, w) to (s, t), which consists of a transformation from the UVW coordinate system to the XYZ coordinate system followed by a perspective projection followed by a transformation to the ST coordinate system.*

transformation. You should confirm the transformation that maps the vector $\mathbf{X}$ representing a point in 3D in homogeneous coordinates to

$$\lambda \begin{bmatrix} \mathcal{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \mathbf{X}$$

represents a Euclidean transformation, but in homogeneous coordinates. It follows that any map of this form is a Euclidean transformation. Because $\mathcal{T}_e$ represents a Euclidean transformation, it must have this form. The exercises explore some properties of $\mathcal{T}_e$.

11.2.4 Camera Intrinsic Parameters

Camera intrinsic parameters must model a possible coordinate transformation in the image plane from projected world coordinates (write (x, y)) to pixel coordinates (write (u, v)), together with a possible change of focal length. This change is caused by the image plane being further away from, or closer to, the focal point. The coordinate transformation is not arbitrary (Figure 11.2). Typically, the origin of the pixel coordinates is usually not at the camera center. Write Δx for the step in the image plane from pixel (i, j) to $(i+1, j)$ and Δy for the step to $(i, j+1)$. These

are vectors parallel to the camera coordinate axes. The vector Δx may not be perpendicular to the vector Δy, causing *skew*. For many cameras, $\|\Delta x\|$ is different from $\|\Delta y\|$ – such cameras have *non-square pixels*, and $\|\Delta x\|/\|\Delta y\|$ is known as the *aspect ratio* of the pixel. Furthermore, $\|\Delta x\|$ is not usually one unit in world coordinates.

There is one tricky point here. Rotating the world about the Z axis has an effect equivalent to rotating the camera coordinate system (Figure **??**). This means we cannot tell whether this rotation is the result of a change in the extrinsics (the world rotated) or the intrinsics (the camera coordinate system rotated). By convention, there is no rotation in the intrinsics, so a pure rotation of the image is always the result of the world rotating.

There are two possible parametrizations of camera intrinsics. Recall f is the focal length of the camera. Write (c'_x, c'_y) for the location of the camera center in pixel coordinates; a for the aspect ratio of the pixels ; and k' for the skew. Then $\mathcal{T}_i$ is parametrized as

$$\begin{bmatrix} \|\Delta x\| & k' & c'_x \\ 0 & \|\Delta y\| & c'_y \\ 0 & 0 & 1/f \end{bmatrix} \equiv \begin{bmatrix} af\|\Delta y\| & fk' & fc'_x \\ 0 & f\|\Delta y\| & fc'_y \\ 0 & 0 & 1 \end{bmatrix}$$

Notice in this case we are distinguishing between scaling resulting from $\|\Delta y\|$ and scaling resulting from the focal length. This is unusual, but can occur. More usual is to conflate these effects and parametrize the intrinsics as

$$\begin{bmatrix} as & k & c_x \\ 0 & s & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where $s = f\|\Delta y\|$, $a = \|\Delta x\|\Delta y$, $k = fk'$, $c_x = fc'_x$, $c_y = fc'_y$.

Remember this: *A general perspective camera can be written in homogeneous coordinates as:*

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \mathcal{T}_i \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathcal{T}_e \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}$$

$$= \begin{bmatrix} as & k & c_x \\ 0 & s & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} \mathcal{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}$$

where $\mathcal{R}$ is a rotation matrix.

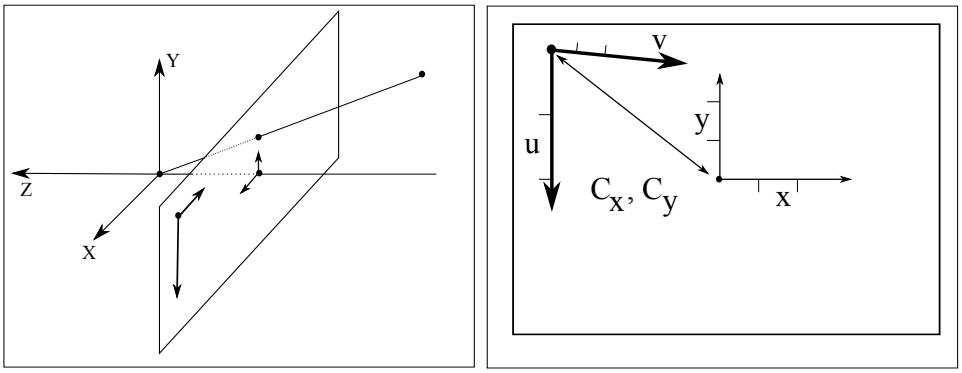

FIGURE 11.2: *The camera reports pixel values in pixel coordinates, which are not the same as world coordinates. The camera intrinsics represent the transformation between world coordinates and pixel coordinates. On the* **left**, *a camera (as in Figure 2.1), with the camera coordinate system shown in heavy lines. On the* **right**, *a more detailed view of the image plane. The camera coordinate axes are marked* (u, v) *and the image coordinate axes* (x, y). *It is hard to determine* f *from the figure, and we will conflate scaling due to* f *with scaling resulting from the change to camera coordinates. The camera coordinate system's origin is not at the camera center, so* (c_x, c_y) *are not zero. I have marked unit steps in the coordinate system with ticks. Notice that the* v*-axis is not perpendicular to the* u*-axis (so* k *-* the skew *- is not zero). Ticks in the* u, v *axes are not the same distance apart as ticks on the* x, y *axes, meaning that* s *is not one. Furthermore,* u *ticks are further apart than* v *ticks, so that* a *is not one.*

By the arguments above, a general orthographic camera transformation can be written as:

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \mathcal{T}_i \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathcal{T}_e \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}$$

PROBLEMS

11.1. We construct the vanishing point of a pair of parallel lines in homogeneous coordinates.

- **(a)** Show that the set of points in homogeneous coordinates in 3D given by $(s, -s, t, s)$ (for s, t parameters) form a line in 3D.
- **(b)** Now image the line $(s, -s, t, s)$ in 3D in a standard perspective camera with focal length 1. Show the result is the line $(s, -s, t)$ in the image plane.
- **(c)** Now image the line $(-s, -s, t, s)$ in 3D in a standard perspective camera with focal length 1. Show the result is the line $(-s, -s, t)$ in the image plane.
- **(d)** Show that the lines $(s, -s, t)$ and $(-s, -s, t)$ intersect in the point $(0, 0, t)$.

11.2. We construct the horizon of a plane for a standard perspective camera with

focal length 1. Write $\mathbf{a} = [a_1, a_2, a_3, a_4]^T$ for the coefficients of the plane, so that for every point $\mathbf{X}$ on the plane we have $\mathbf{a}^T\mathbf{X} = 0$.

(a) Show that the plane given by $\mathbf{u} = [a_1, a_2, a_3, 0]$ is parallel to the plane given by $\mathbf{a}$, and passes through $(0, 0, 0, 1)$.

(b) Write the points on the image plane $(u, v, 1) \equiv (U, V, W)$ in homogeneous coordinates. Show that the horizon of the plane is the set of points $\mathbf{u}$ in the image plane given by $\mathbf{l}^T\mathbf{u} = 0$, where $\mathbf{l} = [a_1, a_2, a_3]^T$.

11.3. A pinhole camera with focal point at the origin and image plane at $z = f$ views two parallel lines $\mathbf{u} + t\mathbf{w}$ and $\mathbf{v} + t\mathbf{w}$. Write $\mathbf{w} = [w_1, w_2, w_3]^T$, etc.

(a) Show that the vanishing point of these lines, on the image plane, is given by $(f\frac{w_1}{w_3}, f\frac{w_2}{w_3})$.

(b) Now we model a family of pairs of parallel lines, by writing $\mathbf{w}(r, s) = r\mathbf{a} + s\mathbf{b}$, for any (r, s). In this model, $\mathbf{u} + t\mathbf{w}(r, s)$ and $\mathbf{v} + t\mathbf{w}(r, s)$ are the pair of lines, and (r, s) chooses the direction. First, show that this family of vectors lies in a plane. Now show that the vanishing point for the (r, s)'th pair is $(f\frac{ra_1+sb_1}{ra_3+sb_3}, f\frac{ra_2+sb_2}{ra_3+sb_3})$.

(c) Show that the family of vanishing points $(f\frac{ra_1+sb_1}{ra_3+sb_3}, f\frac{ra_2+sb_2}{ra_3+sb_3})$ lies on a straight line in the image. Do this by constructing $\mathbf{c}$ such that $\mathbf{c}^T\mathbf{a} = \mathbf{c}^T\mathbf{b} = 0$. Now write $(x(r, s), y(r, s)) = (-f\frac{ra_1+sb_1}{ra_3+sb_3}, -f\frac{ra_2+sb_2}{ra_3+sb_3})$ and show that $c_1x(r, s) + c_2y(r, s) + c_3 = 0$.

11.4. All points on the projective plane with homogeneous coordinates $(U, V, 0)$ lie "at infinity" (divide by zero). As we have seen, these points form a projective line.

(a) Show this line is represented by the vector of coefficients $(0, 0, C)$.

(b) A homography $\mathcal{M} = \left[\mathbf{m}_1^T; \mathbf{m}_2^T; \mathbf{m}_3^T\right]$ is applied to the projective plane. Show that the line whose coefficients are $\mathbf{v}_3$ maps to the line at infinity.

(c) Now write the homography as $\mathcal{M} = \left[\mathbf{m}'_1, \mathbf{m}'_2, \mathbf{m}'_3\right]$ (so $\mathbf{m}'$ are columns). Show that the homography maps the points at infinity to a line given in parametric form as $s\mathbf{m}'_1 + t\mathbf{m}'_2$.

(d) Now write $\mathbf{n}$ for a non-zero vector such that $\mathbf{n}^T\mathbf{m}'_1 = \mathbf{n}^T\mathbf{m}'_2 = 0$. Show that, for any point $\mathbf{x}$ on the line given in parametric form as $s\mathbf{m}'_1 + t\mathbf{m}'_2$, we have $\mathbf{n}^T\mathbf{x} = 0$. Is $\mathbf{n}$ unique?

(e) Use the results of the previous subexercises to show that for any given line, there are some homographies that map that line to the line at infinity.

(f) Use the results of the previous subexercises to show that for any given line, there are some homographies that map the line at infinity to that line.

11.5. We will show that there is no significant difference between choosing a right-handed camera coordinate system and a left-handed camera coordinate system. Notice that, in a right handed camera coordinate system (where the camera looks down the negative z-axis rather than the positive z-axis) the image plane is at $z = -f$.

(a) Show that, in a right-handed coordinate system, a pinhole camera maps

$$(X, Y, Z) \to (-fX/Z, -fY/Z).$$

(b) Show that the argument in the text yields a camera matrix of the form

$$\mathcal{C}'_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{bmatrix}.$$

(c) Show that, if one allows the scale in $\mathcal{T}_i$ to be negative, one could still use

$$\mathcal{C}_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix}$$

as a camera matrix.

CHAPTER 12

Interest Points

One strategy for registering an image to another is to find *interest points* and register those. Interest points have the following important properties:

- It must be possible to find them reasonably reliably, even when image brightness changes.
- It must be possible to *localize* the point (ie tell where the point is) by looking at an image window around the point. For example, a corner can be localized; but a point along a straight edge can't, because sliding a window around the point along the edge leads to a new window that looks like the original.
- The location of the point must be *covariant* under at least some natural image transformations. This means that, if the image is transformed, the point will be found in an appropriate spot in the transformed image. Equivalently, the points "stick to" objects in the image – if the camera moves, the point stays on the object where it was, and so moves in the image. So if, for example, if I_2 is obtained by rotating I_1, then there should be an interest point at each location in I_2 obtained by rotating the position of an interest point in I_1.
- It must be possible to compute a description of the image in the neighborhood of the point, so the point can be matched. Ideally, corresponding points in different images will have similar descriptions, and different points will have different images. To compute this description, we need to be able to construct a neighborhood of the interest point that is covariant. So, for example, if the image is zoomed in, the neighborhood in the image gets bigger; and if it is zoomed out, the neighborhood gets smaller. Using a fixed size neighborhood when the image zooms won't work, because the neighborhood in the zoomed in image will contain patterns that aren't in the neighborhood in the zoomed out image.

These properties are summarized in Figure **??**. The direct constructions for interest points are worth reviewing, because they expose how these properties are achieved. Learned constructions are now competitive with direct constructions, and I describe one in section 32.2.

12.1 DIRECT INTEREST POINT DETECTORS

12.1.1 Finding Corners

Interest points are usually constructed at corners, because they can be localized and are quite easy to find with a straightforward detector. At a corner, we expect two important effects. First, there should be large gradients. Second, in a small neighborhood, the gradient orientation should swing sharply. We can identify corners by looking at variations in orientation within a window. In particular, the

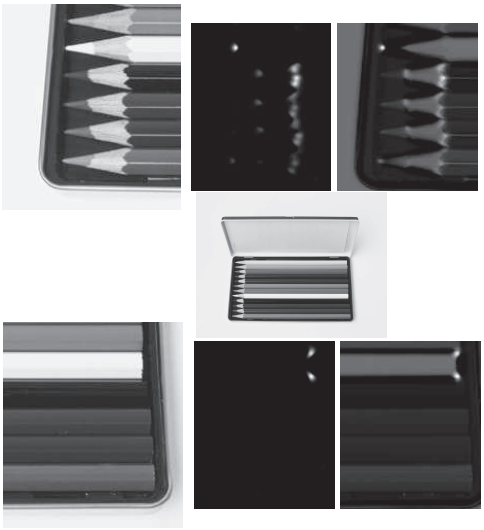

FIGURE 12.1: *The response of the Harris corner detector visualized for two detail regions of an image of a box of colored pencils (***center***).* **Top left***, a detail from the pencil points;* **top center***, the response of the Harris corner detector, where more positive values are lighter. The* **top right** *shows these overlaid on the original image. To overlay this map, we added the images, so that areas where the overlap is notably dark come from places where the Harris statistic is negative (which means that one eigenvalue of* $\mathcal{H}$ *is large, the other small). Note that the detector is affected by contrast, so that, for example, the point of the mid-gray pencil at the top of this figure generates a very strong corner response, but the points of the darker pencils do not, because they have little contrast with the tray. For the darker pencils, the strong, contrasty corners occur where the lead of the pencil meets the wood. The* **bottom** *sequence shows corners for a detail of pencil ends. Notice that responses are quite local, and there are a relatively small number of very strong corners.* Steve Gorton © Dorling Kindersley, used with permission.

matrix

$$\begin{aligned}\mathcal{H} &= \sum_{window} \left\{(\nabla I)(\nabla I)^T\right\} \\ &\approx \sum_{window} \left\{ \begin{array}{cc} (\frac{\partial G_\sigma}{\partial x} * *\mathcal{I})(\frac{\partial G_\sigma}{\partial x} * *\mathcal{I}) & (\frac{\partial G_\sigma}{\partial x} * *\mathcal{I})(\frac{\partial G_\sigma}{\partial y} * *\mathcal{I}) \\ (\frac{\partial G_\sigma}{\partial x} * *\mathcal{I})(\frac{\partial G_\sigma}{\partial y} * *\mathcal{I}) & (\frac{\partial G_\sigma}{\partial y} * *\mathcal{I})(\frac{\partial G_\sigma}{\partial y} * *\mathcal{I}) \end{array} \right\}\end{aligned}$$

gives a good idea of the behavior of the orientation in a window. In a window of constant gray level, both eigenvalues of this matrix are small because all the terms are small. In an edge window, we expect to see one large eigenvalue associated with gradients at the edge and one small eigenvalue because few gradients run in other directions. But in a corner window, both eigenvalues should be large.

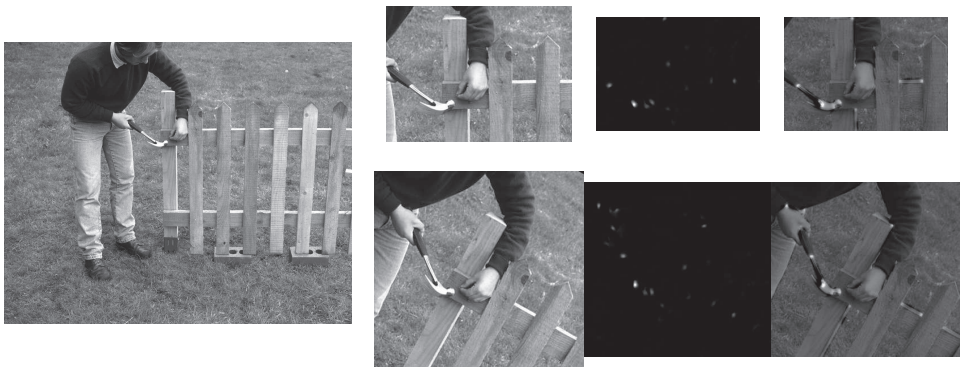

FIGURE 12.2: *The response of the Harris corner detector is unaffected by rotation and translation. The* **top row** *shows the response of the detector on a detail of the image on the* **far left**. *The* **bottom row** *shows the response of the detector on a corresponding detail from a rotated version of the image. For each row, we show the detail window (***left***); the response of the Harris corner detector, where more positive values are lighter (***center***); and the responses overlaid on the image (***right***). Notice that responses are quite local, and there are a relatively small number of very strong corners. To overlay this map, we added the images, so that areas where the overlap is notably dark come from places where the Harris statistic is negative (which means that one eigenvalue of $\mathcal{H}$ is large, the other small). The arm and hammer in the top row match those in the bottom row; notice how well the maps of Harris corner detector responses match, too.*

The *Harris corner detector* looks for local maxima of

$$\det(\mathcal{H}) - k(\frac{\text{trace}(\mathcal{H})}{2})^2$$

where k is some constant [**?**]; we used 0.5 for Figure 12.1. These local maxima are then tested against a threshold. This tests whether the product of the eigenvalues (which is $\det(\mathcal{H})$) is larger than the square of the average (which is $(\text{trace}(\mathcal{H})/2)^2$). Large, locally maximal values of this test function imply the eigenvalues are both big, which is what we want. Figure 12.1 illustrates corners found with the Harris detector. This detector is unaffected by translation and rotation (Figure 12.2).

12.1.2 Building Neighborhoods

There are many ways of representing a neighborhood around an interesting corner. Methods vary depending on what might happen to the neighborhood. In what follows, we will assume that neighborhoods are only translated, rotated, and scaled (rather than, say, subjected to an affine or projective transformation), and so without loss of generality we can assume that the patches are circular. We must estimate the radius of this circle. There is technical machinery available for the neighborhoods that result from more complex transformations, but it is more intricate; see [].

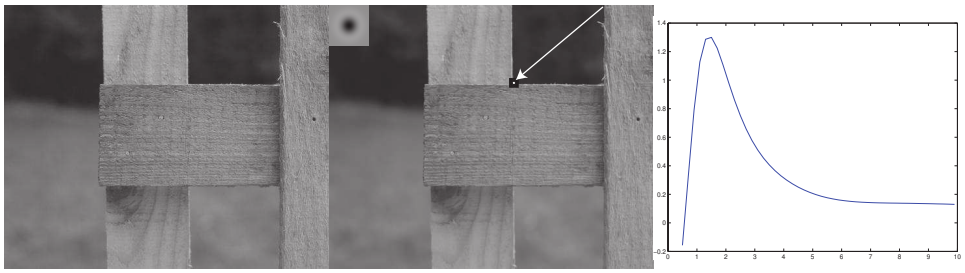

FIGURE 12.3: *The scale of a neighborhood around a corner can be estimated by finding a local extremum, in* scale *of the response at that point to a smoothed Laplacian of Gaussian kernel. On the* **left**, *a detail of a piece of fencing. In the center, a corner identified by an arrow (which points to the corner, given by a white spot surrounded by a black ring).* Overlaid *on this image is a Laplacian of Gaussian kernel, in the* **top right** *corner; dark values are negative, mid gray is zero, and light values are positive. Notice that, using the reasoning of Section 8.3, this filter will give a strong positive response for a dark blob on a light background, and a strong negative response for a light blob on a dark background, so by searching for the strongest response at this point as a function of scale, we are looking for the size of the best-fitting blob. On the* **right**, *the response of a Laplacian of Gaussian* at the location of the corner, *as a function of the smoothing parameter (which is plotted in pixels). There is one extremal scale, at approximately 2 pixels. This means that there is one scale at which the image neighborhood looks most like a blob (some corners have more than one scale).*

To turn a corner into an image neighborhood, we must estimate the radius of the circular patch (equivalently, its scale). The radius estimate should get larger proportionally when the image gets bigger. For example, in a 2x scaled version of the original image, our method should double its estimate of the patch radius. This property helps choose a method. We could center a blob of fixed appearance (say, dark on a light background) on the corner, and then choose the scale to be the radius of the best fitting blob. An efficient way to do this is to use a Laplacian of Gaussian filter.

The *Laplacian* of a function in 2D is defined as

$$(\nabla^2 f)(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}.$$

It is natural to smooth the image before applying a Laplacian. Notice that the Laplacian is a linear operator (if you're not sure about this, you should check), meaning that we could represent taking the Laplacian as convolving the image with some kernel (which we write as K_{∇^2}). Because convolution is associative, we have that

$$(K_{\nabla^2} * *(G_\sigma * *I)) = (K_{\nabla^2} * *G_\sigma) * *I = (\nabla^2 G_\sigma) * *I.$$

The reason this is important is that, just as for first derivatives, smoothing an image and then applying the Laplacian is the same as convolving the image with

the Laplacian of the kernel used for smoothing. Figure 12.3 shows the resulting kernel for Gaussian smoothing; notice that this looks like a dark blob on a light background.

Imagine applying a smoothed Laplacian operator to the image at the center of the patch. Write $\mathcal{I}$ for the image, ∇^2_σ for the smoothed Laplacian operator with smoothing constant σ, $\uparrow_k \mathcal{I}$ for the the image with size scaled by k, (x_c, y_c) for the coordinates of the patch center, and (x_{kc}, y_{kc}) for the coordinates of the patch center in the scaled image. Assume that upscaling is perfect, and there are no effects resulting from the image grid. This is fair because effects will be small for the scales of interest for us. Then, we have

$$(\nabla^2_{k\sigma} \uparrow_k \mathcal{I})(x_c, y_c) = (\nabla^2_\sigma \mathcal{I})(x_{kc}, y_{kc})$$

(this is most easily demonstrated by reasoning about the image as a continuous function, the operator as a convolution, and then using the change of variables formula for integrals). Now choose a radius r for the circular patch centered at (x_c, y_c), such that

$$r(x_c, y_c) = \underset{\sigma}{\text{argmax}} \ \nabla^2_\sigma \mathcal{I}(x_c, y_c)$$

(Figure 12.3). If the image is scaled by k, then this value of r will be scaled by k too, which is the property we wanted. This procedure looks for the scale of the best approximating blob. Notice that a Gaussian pyramid could be helpful here; we could apply the same smoothed Laplacian operator to different levels of a pyramid to get estimates of the scale.

As we have seen, orientation histograms are a natural representation of image patches. However, we cannot represent orientations in image coordinates (for example, using the angle to the horizontal image axis), because the patch we are matching to might have been rotated. We need a reference orientation so all angles can be measured with respect to that reference. A natural reference orientation is the most common orientation in the patch. We compute a histogram of the gradient orientations in this patch, and find the largest peak. This peak is the reference orientation for the patch. If there are two or more peaks of the same magnitude, we make multiple copies of the patch, one at each peak orientation.

12.1.3 Describing Neighborhoods with Orientations

We know the center, radius, and orientation of a set of an image patch, and must now represent it. Orientations should provide a good representation. They are unaffected by changes in image brightness, and different textures tend to have different orientation fields. The pattern of orientations in different parts of the patch is likely to be quite distinctive. Our representation should be robust to small errors in the center, radius, or orientation of the patch, because we are unlikely to estimate these exactly right.

We must build features that can make it obvious what orientations are present, and roughly where they are, but are robust to some rearrangement. One approach is to represent the neighborhood with a histogram of the elements that appear there. This will tell us what is present, but it confuses too many patterns with one another. For example, all neighborhoods with vertical stripes will get mixed

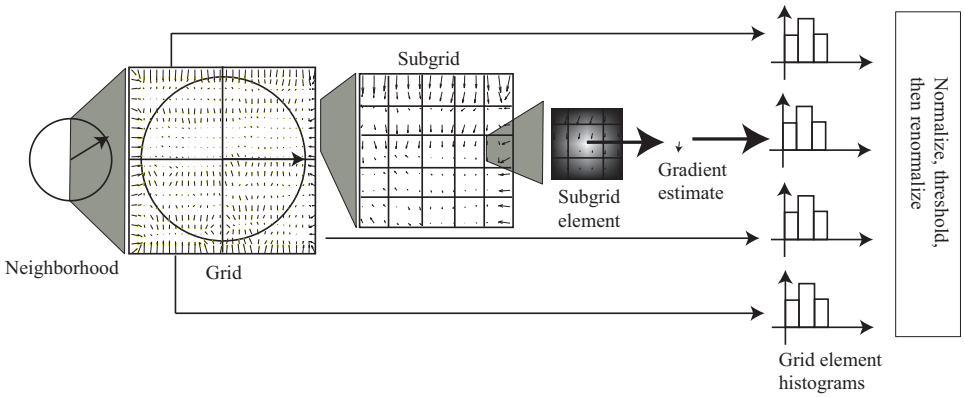

FIGURE 12.4: *To construct a SIFT descriptor for a neighborhood, we place a grid over the rectified neighborhood. Each grid is divided into a subgrid, and a gradient estimate is computed at the center of each subgrid element. This gradient estimate is a weighted average of nearby gradients, with weights chosen so that gradients outside the subgrid cell contribute. The gradient estimates in each subgrid element are accumulated into an orientation histogram. Each gradient votes for its orientation, with a vote weighted by its magnitude and by its distance to the center of the neighborhood. The resulting orientation histograms are stacked to give a single feature vector. This is normalized to have unit norm; then terms in the normalized feature vector are thresholded, and the vector is normalized again.*
TODO: Source, Credit, Permission: SIFTPIC

up, however wide the stripe. The natural approach is to take histograms locally, within subpatches of the neighborhood. This leads to a very important feature construction.

A *SIFT descriptor* (for Scale Invariant Feature Transform) is constructed out of image gradients, and uses both magnitude and orientation. The descriptor is normalized to suppress the effects of change in illumination intensity. The descriptor is a set of histograms of image gradients that are then normalized. These histograms expose general spatial trends in the image gradients in the patch but suppress detail. For example, if we estimate the center, scale, or orientation of the patch slightly wrong, then the rectified patch will shift slightly. As a result, simply recording the gradient at each point yields a representation that changes between instances of the patch. A histogram of gradients will be robust to these changes. Rather than histogramming the gradient at a set of sample points, we histogram local averages of image gradients; this helps avoid noise.

There is now extensive experimental evidence that image patches that match one another will have similar SIFT feature representations, and patches that do not will tend not to.

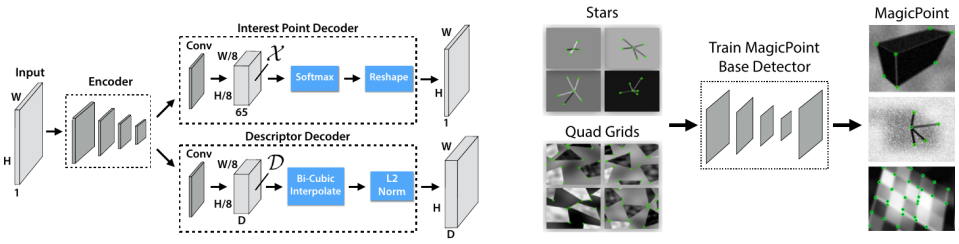

FIGURE 12.5: *SuperPoint uses an encoder with two heads (***left***), one of which predicts the locations of interest points and the other of which predicts a descriptor. The location finder assumes that there is at most one interest point per 8x8 image tile, and predicts which (if any) location is that point. A basic location finder is trained using a cross-entropy loss with a dataset of rendered images where interest point locations are known (***right***).*
TODO: Source, Credit, Permission

12.2 SUPERPOINT: A LEARNED INTEREST POINT DETECTOR

It turns out the list of properties of interest points is crisp enough that one can learn an interest point finder, and learned interest point finders now are dominant. SuperPoint uses a network architecture that is adapted to fast computation of points and descriptors, with a mixture of learned and non-learned components. This is trained in a series of steps. The first builds an elementary interest point finder. The second uses a clever trick with image transformations to significantly improve the interest point finder. The third refines point positions and descriptors with a matching loss.

12.2.1 Network Architecture

First, pass the image through an encoder, which encodes the image with series of convolutional layers, non-linear layers, and three 2×2 downsampling layers, so that it takes an $H \times W$ image and produces an $H/8 \times W/8 \times 256$ feature block. This block goes to two heads. One finds interest points, the other describes them. The interest point finder is trained discriminatively, by dividing the image into a grid of $H/8 \times W/8$ tiles (each tile is 8×8 pixels). Now assume there is at most one interest point in any tile. A 65 dimensional one-hot vector encodes where the interest point is if there is one (there are 64 locations for the point, and the last component is one if there isn't a point). The interest point finder maps the original block to an $H/8 \times W/8 \times 65$ block, which is passed through a softmax. Reshaping this with a fixed reshaping procedure gives the predicted location of the interest point. The interest point describer maps the original block to an $H/8 \times W/8 \times 256$ block. This is upsampled using a bicubic interpolation procedure (Section 32.2), and the predicted vector at each location is normalized to a unit vector.

TODO: Brief description of bicubic interpolation somewhere

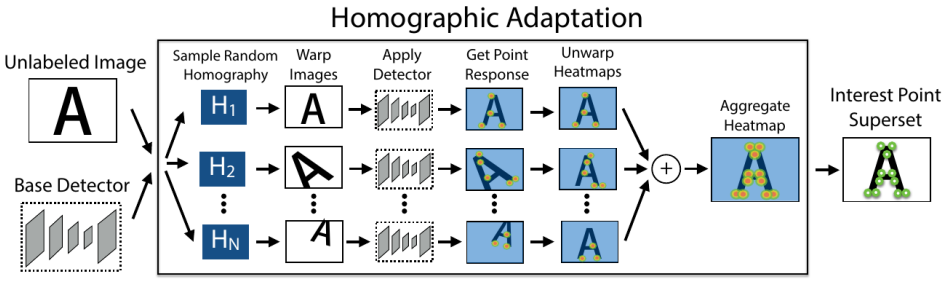

FIGURE 12.6: *The basic location finder of Figure 9 can be significantly improved by exploiting the constraint that interest point predictions should be covariant. The response of the finder to a transformed image, which is a heatmap, should be a transformed version of the response to the original image. Equivalently, apply the finder to a transformed image, and the inverse of the transformation to the resulting heatmap – that heatmap should be the same as the one the detector produces from the original image. This means that a composite finder can be built out of the the basic location finder by predicting heatmaps from images transformed with random (but carefully chosen) homographies, transforming the heatmaps back to the original image frame, then averaging them. Training this composite finder improves the original basic finder, without requiring real data.*
TODO: Source, Credit, Permission

12.2.2 Finding Interest Points

Generating a large number of relatively simple images with known interest point locations is easy. Use a simple computer graphics program to render collections of polygons; each vertex is an interest point. If any image has more than one interest points in one tile, discard all but one at random. We now have a labelled dataset of images (the labels are interest point locations), and a basic detector can be trained with this.

An interest point detector should be covariant under homographies – the interest points for a transformed image should be obtained by transforming the interest points of the original image. This likely won't be a property of the basic interest point detector, but it can be self-supervised very strongly using this idea. Write $f(\mathcal{I}, \theta)$ for the output of the interest point detector with parameters θ applied to the image $\mathcal{I}$ (this is a *heat map* – at every pixel location, there is a value giving the probability of an interest point at that location), and $\mathcal{H}(\mathcal{I})$ for the result of applying a homography to $\mathcal{I}$. The output of the detector can be thought of as an image, so a homography can be applied to it. Covariance means that the heat map $\mathcal{H}^{-1}(f(\mathcal{H}(\mathcal{I}), \theta))$ should be the same as $f(\mathcal{I}, \theta)$, at least for reasonable choices of $\mathcal{H}$. For each of the training images above, choose a collection of N homographies at random (taking care with cropping, etc. – details in []), and train the detector which produces the heat map

$$\frac{1}{N} \sum_i \mathcal{H}_i^{-1}(f(\mathcal{H}(\mathcal{I}), \theta)).$$

Training like this has quite strong effects on θ because the detector receives gradient if (say) an interest point is detected in the wrong place in a (say) rotated version of the original image. It is also an extremely efficient use of data.

12.2.3 Refining Detection and Learning to Describe

The refined detector can now be trained to improve interest point detections and to produce descriptions. Take a synthetic image with interest points known and apply a homography. The interest points in the result should be close to those predicted by applying the homography to the interest points in the original image. This property can be imposed with a cross-entropy loss between $\mathcal{H}f(\mathcal{I}, \theta)$ and $f(\mathcal{H}(\mathcal{I}), \theta)$.

Corresponding points in $\mathcal{I}$ and $\mathcal{H}(\mathcal{I})$ should have similar descriptors and pairs of points that don't correspond should have different descriptors. It is easier to impose this on tiles than points. For every pair of tiles, where one comes from $\mathcal{I}$ and the other from $\mathcal{H}(\mathcal{I})$, say the pair corresponds if there is some interest point in the first that maps to a point in the second. Otherwise, the pair does not correspond. Recall that the descriptors are computed on a coarse grid where each location corresponds to a tile (and are then upsampled). Write $\mathbf{d}(t)$ for the descriptor of a tile, and so on. The matching loss is a hinge loss that ensures that, if tiles t and t' correspond, then $\mathbf{d}^T(t)\mathbf{d}(t')$ is positive and greater than some margin, and if they do not, it is negative and less than some margin.

Resources: Interest Points *A pretrained version of SuperPoint can be found at* `https://github.com/magicleap/SuperPointPretrainedNetwork`. *There are implementations for TensorFlow () and PyTorch (). HPatches is an evaluation dataset (at* `https://github.com/hpatches/hpatches-dataset`*) which comes with evaluation protocols and benchmarks (at* `https://github.com/hpatches/hpatches-benchmark`*). OpenCV provides an implementation of the Harris corner detector, and procedures to compute SIFT descriptors.*

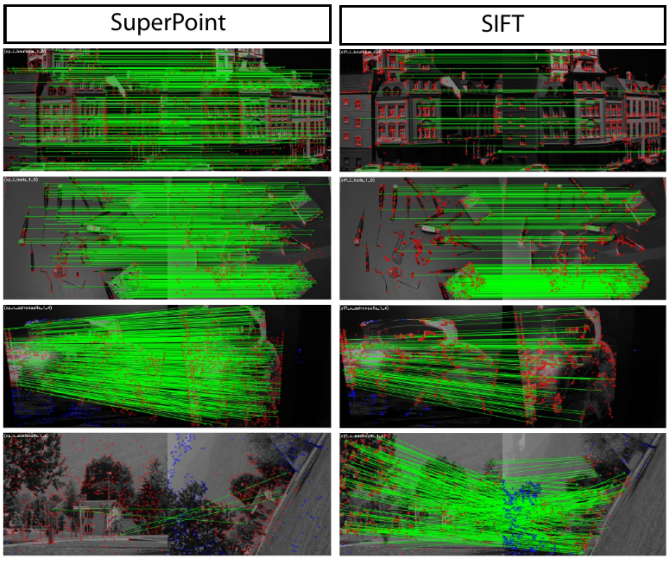

FIGURE 12.7: *SuperPoint produces many good interest point locations together with descriptors that are distinctive.* **Left** *shows SuperPoint detections and matches for four image pairs, and* **right** *for a SIFT based matcher. The images are transformed with a known homography (red dots are detected interest points; blue dots are detected interest points that are outside the field of view of the corresponding image, and so could not have a match; green lines indicate matches). Generally, SuperPoint produces large numbers of interest points that match well. The original SuperPoint is trained with relatively small image rotations, because big rotations are less common in practice, and so handles large image rotations poorly compared to a SIFT based matcher (fourth row).*
TODO: Source, Credit, Permission

CHAPTER 13

Registration

Computing a transformation that aligns an image or a depth map or a set of images with another such is generally known as *registration*. One approach to registration is to abstract the image (etc.) as a set of points, yielding the following general problem. Assume we know a set of N reference points in d dimensions. We observe M points in d dimensions, and these observed M points are obtained by transforming the reference points with some transformation and adding noise, then dropping some points and including some pure noise points. The two sets of points are often referred to as *point clouds*. We want to determine the transformation from the two point clouds.

This problem occurs in a wide range of practical applications. As we shall see, calibrating a camera involves solving a version of this problem (Section 32.2). Determining where you are in a known map very often involves solving a version of this problem. Imagine, for example, a camera looking directly downwards from an aircraft flying at fixed height. The image in the camera translates and rotates as the aircraft moves. If we can compute the transformation from image i to image $i+1$, we can tell how the aircraft has moved. Another useful case occurs when we have a depth map of a known object and want to compute the *pose* of the object (its position and orientation in the frame of the depth sensor). We could do so by having reference points on the object, finding interest points in the depth map, then solving for a transformation that maps reference points to depth points.

How one approaches this class of problem depends on three important factors.

- **Correspondence:** if it is known *which* observation corresponds to *which* reference point, the problem is relatively straightforward to solve (unless there are unusual noise effects). This case is uncommon, but does occur. In robotics, *beacons* are objects that identify themselves (perhaps by wearing a barcode; by transmitting some code; by a characteristic pattern) and can be localized. They are useful, precisely because they yield correspondence and so simplify computing the transformation. If correspondence is not known, which is the usual case, computing the transformation becomes rather harder.

- **Transformation:** there are closed form solutions for known correspondence and Euclidean or affine transformations. Homographies (and higher dimensional analogs) do not admit closed form transformations.

- **Noise:** computing a transformation can become very hard if many of the observations do not come from reference points, if many of the reference points are dropped, or if some observations are subject to very large noise effects.

13.1 REGISTRATION WITH KNOWN CORRESPONDENCE AND GAUSSIAN NOISE

13.1.1 Affine Transformations and Gaussian Noise

In the simplest case, the correspondence is known – perhaps the reference points are beacons – and the only noise is Gaussian (so $N = M$). Write $\mathbf{x}_i$ for the i'th observation and $\mathbf{y}_i$ for the i'th reference point. We will assume the noise is isotropic, which is by far the most usual case. Once you have followed this derivation, you will find it easy to incorporate a known covariance matrix. We have

$$\mathbf{x}_i = \mathcal{M}\mathbf{y}_i + \mathbf{t} + \xi_i$$

where ξ_i is the value of a normal random variable with mean $\mathbf{0}$ and covariance matrix $\Sigma = \sigma^2\mathcal{I}$. A natural procedure to estimate $\mathcal{M}$ and $\mathbf{t}$ is to maximize the likelihood of the noise. Because it will be useful later, we assume that there is a weight w_i for each pair, so the negative log-likelihood we must minimize is proportional to

$$\sum_i w_i \left(\mathbf{x}_i - \mathcal{M}\mathbf{y}_i - \mathbf{t}\right)^T \left(\mathbf{x}_i - \mathcal{M}\mathbf{y}_i - \mathbf{t}\right)$$

(the constant of proportionality is σ^2, which doesn't affect the optimization problem). The gradient of this cost with respect to $\mathbf{t}$ is

$$-2\sum_i w_i \left(\mathbf{x}_i - \mathcal{M}\mathbf{y}_i - \mathbf{t}\right)$$

which vanishes at the solution. In turn, if $\sum_i w_i\mathbf{x}_i = \sum_i w_i\mathcal{M}\mathbf{y}_i$, $\mathbf{t} = \mathbf{0}$. One straightforward way to achieve this is to ensure that both the observations and the reference points have a center of gravity at the origins. Write

$$\mathbf{c}_x = \frac{\sum_i w_i\mathbf{x}_i}{\sum_i w_i}$$

for the center of gravity of the observations (etc.) Now form

$$\mathbf{u}_i = \mathbf{x}_i - \mathbf{c}_x \text{ and } \mathbf{v}_i = \mathbf{y}_i - \mathbf{c}_y$$

and if we use $\mathbf{u}_i$ as observations and $\mathbf{v}_i$ as reference points, then the translation will be zero. In turn, the translation from the original reference points to the original observations is $\mathbf{c}_x - \mathbf{c}_y$.

We obtain $\mathcal{M}$ by minimizing

$$\sum_i w_i \left(\mathbf{u}_i - \mathcal{M}\mathbf{v}_i\right)^T \left(\mathbf{u}_i - \mathcal{M}\mathbf{v}_i\right).$$

Now write $\mathcal{W} = \mathsf{diag}\left([w_1, \ldots, w_N]\right)$, $\mathcal{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_N]$ (and so on). You should check that the objective can be rewritten as

$$\mathsf{Tr}\left(\mathcal{W}(\mathcal{U} - \mathcal{M}\mathcal{V})^T(\mathcal{U} - \mathcal{M}\mathcal{V})\right).$$

Now the trace is linear; $\mathcal{U}^T\mathcal{U}$ is constant; and we can rotate matrices through the trace (Section 32.2). This means the cost is equivalent to

$$\mathsf{Tr}\left(-2\mathcal{M}\mathcal{V}\mathcal{U}^T + \mathcal{M}^T\mathcal{M}\mathcal{V}\mathcal{V}^T\right)$$

which will be minimized when

$$\mathcal{M}\mathcal{V}\mathcal{W}\mathcal{V}^T = \mathcal{V}\mathcal{W}\mathcal{U}^T$$

(which you should check). Many readers will recognize a least squares solution here. The trace isn't necessary here, but it's helpful to see an example using the trace, because it will be important in the next case.

13.1.2 Euclidean Motion and Gaussian Noise

One encounters affine transformations relatively seldom in practice, though they do occur. Much more interesting is the case where the transformation is Euclidean. The least squares solution above isn't good enough, because the $\mathcal{M}$ obtained that way won't be a rotation matrix. But we can obtain a least squares solution with a rotation matrix, using a neat trick. We adopt the notation of the previous section, and change coordinates from $\mathbf{x}_i$ to $\mathbf{u}_i$ as above to remove the need to estimate translation.

We must choose $\mathcal{R}$ to minimize

$$\sum_i w_i(\mathbf{u}_i - \mathcal{R}\mathbf{v}_i)^T(\mathbf{u}_i - \mathcal{R}\mathbf{v}_i).$$

This can be done in closed form (a fact you should memorize). Equivalently, we must minimize

$$\begin{aligned}\sum_i w_i(\mathbf{u}_i - \mathcal{R}\mathbf{v}_i)^T(\mathbf{u}_i - \mathcal{R}\mathbf{v}_i) &= \mathsf{Tr}\left(\mathcal{W}(\mathcal{U} - \mathcal{R}\mathcal{V})(\mathcal{U} - \mathcal{R}\mathcal{V})^T\right) \\ &= \mathsf{Tr}\left(-2\mathcal{U}\mathcal{W}\mathcal{V}^T\mathcal{R}^T\right) + K \\ &\quad \text{(because } \mathcal{R}^T\mathcal{R} = \mathcal{I}\text{)} \\ &= -2\mathsf{Tr}\left(\mathcal{R}\mathcal{U}\mathcal{W}\mathcal{V}^T\right)\end{aligned}$$

Now we compute an SVD of $\mathcal{U}\mathcal{V}^T$ to obtain $\mathcal{U}\mathcal{W}\mathcal{V}^T = \mathcal{A}\mathcal{S}\mathcal{B}^T$ (where $\mathcal{A}$, $\mathcal{B}$ are orthonormal, and $\mathcal{S}$ is diagonal – Section 32.2 if you're not sure). Now $\mathcal{B}^T\mathcal{R}\mathcal{A}$ is orthonormal, and we must maximize $\mathsf{Tr}\left(\mathcal{B}^T\mathcal{R}\mathcal{A}\mathcal{S}\right)$, meaning $\mathcal{B}^T\mathcal{R}\mathcal{A} = \mathcal{I}$ (check this if you're not certain), and so $\mathcal{R} = \mathcal{B}\mathcal{A}^T$.

Procedure: 13.1 *Weighted Least Squares for Euclidean Transformations*

We have N reference points $\mathbf{x}_i$ whose location is measured in the agent's coordinate system. Each corresponds to a point in the world coordinate system with known coordinates $\mathbf{y}_i$, and the change of coordinates is a Euclidean transformation (rotation $\mathcal{R}$, translation $\mathbf{t}$). For each $(\mathbf{x}_i, \mathbf{y}_i)$ pair, we have a weight w_i. We wish to minimize

$$\sum_i w_i(\mathbf{x}_i - \mathcal{R}\mathbf{y}_i - \mathbf{t})^T(\mathbf{x}_i - \mathcal{R}\mathbf{y}_i - \mathbf{t})$$

Write

$$\begin{aligned}\mathbf{c}_x &= \frac{\sum_i w_i\mathbf{x}_i}{\sum_i w_i} \\ \mathbf{c}_y &= \frac{\sum_i w_i\mathbf{y}_i}{\sum_i w_i} \\ \mathbf{u}_i &= \mathbf{x}_i - \mathbf{c}_x \\ \mathbf{v}_i &= \mathbf{y}_i - \mathbf{c}_y\end{aligned}$$

Then the least squares estimate $\hat{\mathbf{t}}$ of $\mathbf{t}$ is

$$\hat{\mathbf{t}} = \mathbf{c}_x - \mathbf{c}_y$$

Write $\mathcal{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N]$ (etc); $\mathcal{W} = \text{diag}(w_1, \ldots, w_N)$; and $\text{SVD}(\mathcal{U}\mathcal{S}\mathcal{V}) = \mathcal{A}\Sigma\mathcal{B}^T$. The least squares estimate $\hat{\mathcal{R}}$ is

$$\hat{\mathcal{R}} = \mathcal{B}\mathcal{A}^T$$

13.1.3 Homographies and Gaussian Noise

We now work with $d = 2$, and allow the transformation to be a homography. Solving for a homography requires solving an optimization problem, but estimating a homography from data is useful, and relatively easy to do. Furthermore, we can't recover the translation component from centers of gravity (exercises **TODO:** homography exercise). In all cases of interest, the points $\mathbf{x}_i$ and $\mathbf{y}_i$ will be supplied in affine coordinates, rather than homogeneous coordinates, and we convert to homogeneous coordinates by attaching a 1, as before. Write m_{ij} for the i, j'th element of matrix $\mathcal{M}$. In affine coordinates, a homography $\mathcal{M}$ will map $\mathbf{y}_i = (y_{i,x}, y_{i,y})$ to $\mathbf{x}_i = (x_{i,x}, x_{i,y})$ where

$$x_{i,x} = \frac{m_{11}y_{i,x} + m_{12}y_{i,y} + m_{13}}{m_{31}x_{i,x} + m_{32}x_{i,y} + m_{33}} \text{ and } x_{i,y} = \frac{m_{21}y_{i,x} + m_{22}y_{i,y} + m_{23}}{m_{31}x_{i,x} + m_{32}x_{i,y} + m_{33}}$$

Write $\mathcal{M}(\mathbf{y})$ for the result of applying the homography to $\mathbf{y}$, *in affine coordinates*. In most cases of interest, the coordinates of the points are not measured precisely, so

we observe $\mathbf{x}_i = \mathcal{M}(\mathbf{y}_i) + \xi_i$, where ξ_i is some noise vector drawn from an isotropic normal distribution with mean $\mathbf{0}$ and covariance Σ.

The error will be in affine coordinates – for example, in the image plane – which justifies working in affine rather than homogeneous coordinates. Again, we assume that the noise is isotropic, and so that $\Sigma = \sigma^2\mathcal{I}$. The homography can be estimated by minimizing the negative log-likelihood of the noise, so we must minimize

$$\sum_i w_i \xi_i^T \xi_i$$

where

$$\xi_i = \begin{bmatrix} x_{i,x} - \frac{m_{11}y_{i,x}+m_{12}y_{i,y}+m_{13}}{m_{31}y_{i,x}+m_{32}y_{i,y}+m_{33}} \\ x_{i,y} - \frac{m_{21}y_{i,x}+m_{22}y_{i,y}+m_{23}}{m_{31}y_{i,x}+m_{32}y_{i,y}+m_{33}} \end{bmatrix}$$

using standard methods (Levenberg-Marquardt is favored; Chapter 32.2). This approach is sometimes known as *maximum likelihood* . Experience teaches that this optimization is not well behaved without a strong start point.

There is an easy construction for a good start point. Notice that the equations for the homography mean that

$$x_{i,x}(m_{31}y_{i,x} + m_{32}y_{i,y} + m_{33}) - m_{11}y_{i,x} + m_{12}y_{i,y} + m_{13} = 0$$

and

$$x_{i,y}(m_{31}y_{i,x} + m_{32}y_{i,y} + m_{33}) - m_{21}y_{i,x} + m_{22}y_{i,y} + m_{23} = 0$$

so each corresponding pair of points $\mathbf{x}_i$, $\mathbf{y}_i$ yields two *homogeneous* linear equations in the coefficients of the homography. They are homogeneous because scaling $\mathcal{M}$ doesn't change what it does to points (check this if you're uncertain). If we obtain sufficient points, we can solve the resulting system of homogeneous linear equations. Four point correspondences yields an unambiguous solution; more than four – which is better – can be dealt with by least squares (exercises **TODO:** fourpoint homography). The resulting estimate of $\mathcal{M}$ has a good reputation as a start point for a full optimization.

Procedure: 13.2 *Estimating a Homography from Data*

Given N known source points $\mathbf{y}_i = (y_{i,x}, y_{i,y})$ in affine coordinates and N corresponding target points $\mathbf{x}_i$ with measured locations $(x_{i,x}, x_{i,y})$ and where measurement noise has zero mean and covariance $\Sigma = \sigma^2\mathcal{I}$, estimate the homography $\mathcal{M}$ with i, j'th element m_{ij} by minimizing:

$$\sum_i \xi_i^T \xi_i$$

where

$$\xi = \begin{bmatrix} x_{i,x} - \frac{m_{11}y_{i,x}+m_{12}y_{i,y}+m_{13}}{m_{31}y_{i,x}+m_{32}y_{i,y}+m_{33}} \\ x_{i,y} - \frac{m_{21}y_{i,x}+m_{22}y_{i,y}+m_{23}}{m_{31}y_{i,x}+m_{32}y_{i,y}+m_{33}} \end{bmatrix}$$

Obtain a start point by as a least-squares solution to the set of homogeneous linear equations

$$x_{i,x}(m_{31}y_{i,x} + m_{32}y_{i,y} + m_{33}) - m_{11}y_{i,x} + m_{12}y_{i,y} + m_{13} = 0$$

and

$$x_{i,y}(m_{31}y_{i,x} + m_{32}y_{i,y} + m_{33}) - m_{21}y_{i,x} + m_{22}y_{i,y} + m_{23} = 0.$$

13.1.4 Projective Transformations and Gaussian Noise

A *projective transformation* is the analogue of a homography for higher dimensions. In affine coordinates, a projective transformation $\mathcal{M}$ will map $\mathbf{y}_i = (y_{i,1}, \ldots, y_{i,d})$ to $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,d})$ where

$$x_{i,1} = \frac{m_{11}y_{i,1} + \ldots + m_{1d}y_{i,d} + m_{1(d+1)}}{m_{(d+1)1}y_{i,1} + \ldots + m_{(d+1)d}y_{i,d} + m_{(d+1)(d+1)}}$$

and

$$x_{i,d} = \frac{m_{d1}y_{i,1} + \ldots + m_{dd}y_{i,d} + m_{d(d+1)}}{m_{(d+1)1}y_{i,1} + \ldots + m_{(d+1)d}y_{i,d} + m_{(d+1)(d+1)}}$$

Estimating this transformation follows the recipe for a homography, but there are now more parameters. I have put the result in a box, below.

Procedure: 13.3 *Estimating a Projective Transformation from Data*

Given N known source points $\mathbf{y}_i = (y_{i,1}, \ldots, y_{i,d})$ in affine coordinates and N corresponding target points $\mathbf{x}_i$ with measured locations $(x_{i,1}, \ldots, x_{i,d})$ and where measurement noise has zero mean and is isotropic, the homography $\mathcal{M}$ with i, j'th element m_{ij} by minimizing:

$$\sum_i \xi_i^T \Sigma^{-1} \xi_i$$

where

$$\xi_i = \begin{bmatrix} x_{i,1} - \frac{m_{11}y_{i,1}+\ldots+m_{1d}y_{i,d}+m_{1(d+1)}}{m_{(d+1)1}x_{i,1}+\ldots+m_{(d+1)d}x_{i,d}+m_{(d+1)(d+1)}} \\ \ldots \\ x_{i,d} - \frac{m_{d1}y_{i,1}+\ldots+m_{dd}y_{i,d}+m_{d(d+1)}}{m_{(d+1)1}x_{i,1}+\ldots+m_{(d+1)d}x_{i,d}+m_{(d+1)(d+1)}} \end{bmatrix}$$

Obtain a start point by as a least squares solution to the set of homogeneous linear equations

$$\begin{aligned} 0 &= x_{i,1}(m_{(d+1)1}y_{i,1} + \ldots + m_{(d+1)d}y_{i,d} + m_{(d+1)(d+1)}) - \\ &\quad m_{11}y_{i,1} + \ldots + m_{1d}y_{i,d} + m_{1(d+1)} \\ &\ldots \\ 0 &= x_{i,d}(m_{(d+1)1}y_{i,1} + \ldots + m_{(d+1)d}y_{i,d} + m_{(d+1)(d+1)}) - \\ &\quad m_{(d+1)1}x_{i,1} + \ldots + m_{(d+1)d}x_{i,d} + m_{(d+1)(d+1)} \end{aligned}$$

13.2 UNKNOWN CORRESPONDENCE

13.2.1 Unknown Correspondence and ICP

Now assume correspondences are not known, and some reference (resp. observed) points may not even have corresponding observed (resp. reference) points. We have N reference points $\mathbf{y}_i$ and M observed points $\mathbf{x}_i$. For the moment, we will assume that all weights w_i are 1. A straightforward, and very effective, recipe for registering the points is *iterative closest points* or *ICP*. The key insight here is that, if the transformation is very close to the identity, then the $\mathbf{y}_{c(i)}$ that corresponds to $\mathbf{x}_i$ should be the closest reference point to $\mathbf{x}_i$. This finding the closest reference point to each measurement and computing the transformation using that correspondence. But the transformation might not be close to the identity, and so the correspondences might change. We could repeat the process until they stop changing.

Formally, start with a transformation estimate $\mathcal{T}_1$, a set of $\mathbf{m}_i^{(1)} = \mathcal{T}^{(1)}(\mathbf{y}_i)$ and then repeat two steps:

- **Estimate correspondences** using the transformation estimate. Then, for each $\mathbf{x}_i$, we find the closest $\mathbf{m}^{(n)}$ (say $\mathbf{m}_c^{(n)}$); then $\mathbf{x}_i$ corresponds to $\mathbf{m}_{c(i)}^{(n)}$.

- **Estimate a transformation** $\mathcal{T}^{(n+1)}$ using the corresponding pairs. This maps each $\mathbf{m}_{c(i)}^{(n)}$ to its corresponding $\mathbf{x}_i$.

These steps are repeated until convergence, which can be tested by checking if the correspondences don't change or if $\mathcal{T}^{(n+1)}$ is very similar to the identity. The require transformation is then

$$\mathcal{T}^{(n+1)} \circ \mathcal{T}^{(n)} \circ \ldots \mathcal{T}^{(1)}$$

There are a number of ways in which this very useful and very general recipe can be adapted. First, if there is any description of the points available, it can be used to cut down on correspondences (so, for example, we match only red points to red points, green points to green points, and so on). Second, finding an exact nearest neighbor in a large point cloud is hard and slow, and we might need to subsample the point clouds or pass to approximate nearest neighbors (more details below). Third, points that are very far from the nearest neighbor might cause problems, and we might omit them (again, more details below).

13.2.2 ICP and Sampling

One particularly useful application of ICP occurs when one wishes to register a mesh to a set of points. For example, you might want to register a cloud of measured points to a mesh model of an object built using a CAD modelling system. A natural procedure is to sample points on the mesh model to get a point cloud, then treat the problem using ICP. Another useful application is when one has two mesh models, where the triangulation of the meshes might not be the same. In this case, you could sample both meshes to end up with two point clouds, then register the point clouds. How one samples the mesh or meshes is important.

The ICP recipe becomes difficult to apply to point clouds when M or N are very large. One obvious strategy to control this problem applies when something else – say, a color measurement – is known about each point. For example, we might get such data by using a range camera aligned with a conventional camera, so that every point in the depth map comes with a color. When extra information is available, one searches only compatible pairs for correspondences.

Large point clouds are fairly common in autonomous vehicle applications. For example, the measurements might be LIDAR measurements of some geometry. It is quite usual now to represent that geometry with another, perhaps enormous, point cloud, which you could think of as a map. Registration would then tell the vehicle where it was in the map. Notice that in this application, there is unlikely to be a measurement that exactly corresponds to each reference point. Instead, when the registration is correct, every $\mathbf{x}_i$ is very close to some transformed $\mathbf{y}_i$, so a least squares estimate is entirely justified. In cases like this, one can subsample the reference point cloud, the measurement point cloud, or both.

The sampling procedure depends on the application, and can have significant effects. For example, imagine we are working with LIDAR on a vehicle which is currently in an open space next to a wall (Figure **??**). There will be many returns from the wall, and likely few from the open space. Uniformly sampled measurements would still have many returns from the wall, and few from the open space. This

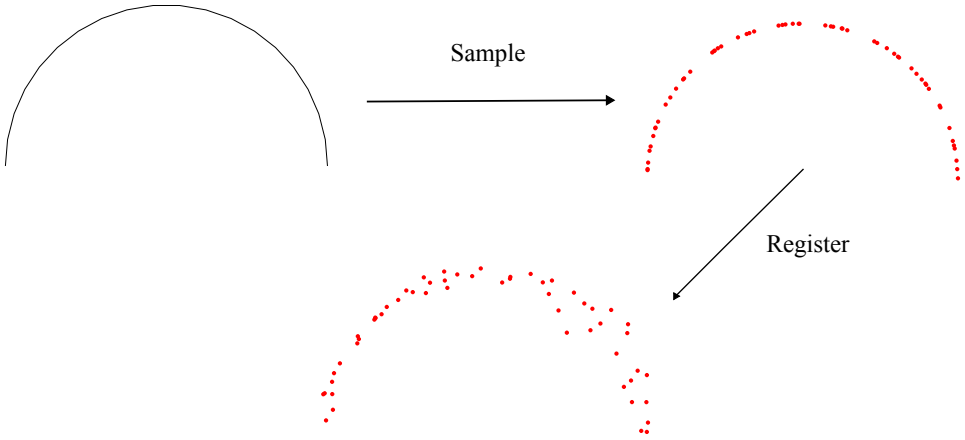

FIGURE 13.1: *In many problems, one has to register a mesh – which might come from a CAD model – to a set of measurements – which might come from LIDAR or from a range camera.* **Top left:** *shows a view of a very simple 1D mesh, in 2D. Registering this mesh to a set of measurements* **bottom** *is a straightforward application of ICP. One samples the mesh* **top left**, *then registers this set of points to the measurements.*

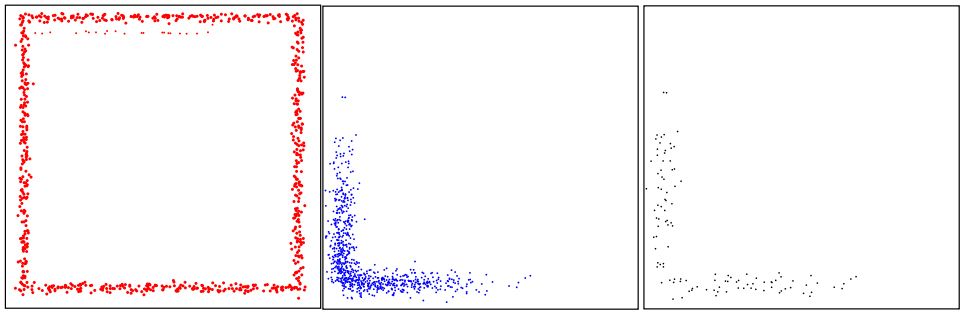

FIGURE 13.2: *On the* **left** *a map of a simple arena, represented as a point cloud. Such a map could be obtained by registering LIDAR measurements to one another. A LIDAR or depth sensor produces measurements in the sensor's coordinate system, and registering these measurements to the map will reveal where the sensor is. However, the sensor may measure points more densely at some positions than at others.* **Left** *shows such a measurement; note the heavy sampling of points near the corner and the light sampling on the edges. This can bias the registration, because the large number of points near the corner mean that the registration error consists mostly of errors from these points. It can also create significant computational problems, because finding the closest points will become slower as the number of points increases. A stratified sample of the measurements (***right***) is obtained by dividing the plane (in this case) into cells of equal area (usually a grid), then resampling the measurements at random* so there are no more than a fixed number of samples in each box. *Such a sample can both reduce bias and improve the speed of registration.* **TODO:** Source, Credit, Permission

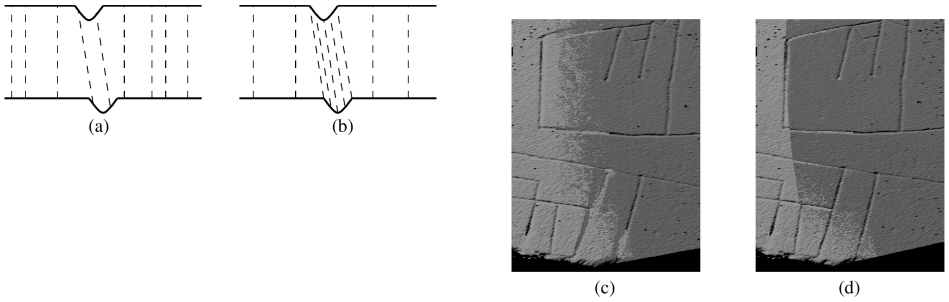

FIGURE 13.3: *The sample of points used in registration can be biased in useful ways. For example, (a) shows a cross section of a flat surface with a small groove (***above***) which needs to be registered to a similar surface (***below***). If point samples are drawn on the surface at random, then there will be few samples in the groove; the dashed lines indicate correspondences. In turn, the registration will be poor, because the surfaces can slide on one another. In (b), the samples have been drawn so that normal directions are evenly represented in the samples. Notice this means more samples concentrated in the groove, and fewer on the flat part. As a result, the surface is less free to slide, and the registration improves.*
TODO: what do c and d show? **TODO:** Source, Credit, Permission

could bias the estimate of the vehicle's pose. A better alternative would be to build a *stratified sample* by breaking the space around the vehicle into blocks of fixed size, then choosing uniformly at random a fixed number of samples in each block. In this scheme, the wall would be undersampled, and the open space would be oversampled, somewhat resolving the bias.

Another stratified sampling strategy is to ensure that surface normal directions are evenly represented in the samples. Make an estimate of a surface normal at each point (for example, by fitting a plane to the point and some of its nearest neighbors). Now break the unit sphere, which encodes the surface normals, into even cells, and sample the points so that each cell has the same number of samples. This approach is particularly useful when we are trying to register flat surfaces with small relief details on them (Figure **??**).

13.2.3 ICP: Finding Nearest Neighbors

Finding the exact nearest neighbor of a query point in a large collection of reference points is more difficult than most people realize (one can beat linear search, but by only a very small factor []). However, finding a point that has high probability of being almost as close as the nearest neighbor (an *approximate nearest neighbor*) can be done rather fast using a variety of approximation schemes []. It is usual to substitute an approximate nearest neighbor, found using a k-d tree (eg []).

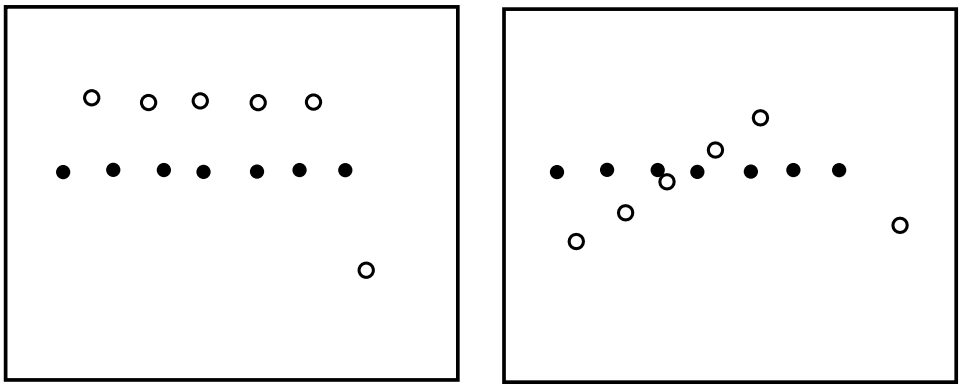

FIGURE 13.4: *Significant registration errors can be caused by just one point that is in the wrong place. On the* **left***, a set of empty points must be registered to a set of filled points. Notice that one empty point is badly out of place. An ideal registration would ignore this, and put the approximate line of empty points on the line of filled points. On the* **right***, the registration that actually results. The square of a large number is very large, meaning the minimum of the squared error isn't where you might think; reducing the large offset entirely justifies a set of medium sized errors. Points that lie significantly far from their "natural" positions are often known as* outliers.

Resources: ICP **TODO:** *ICP Resources*

13.3 NOISE THAT ISN'T GAUSSIAN: ROBUSTNESS AND IRLS

In our examples, if we assume the noise is normal and isotropic, the squared error is reasonably described as negative log-likelihood. But in some cases, even when the measurements and the reference points are properly aligned, some measurement points may lie quite far from the closest reference point. One reason is pure error. Effects like scattering from rain or translucency can cause LIDAR or depth sensors to report measurements that are quite different from the actual geometry. Another is overhangs, which occur when either the reference or measured set contains points representing geometry that isn't in the other set. In this case, some points from one set should be far away from the closest point in the other set. Each of these effects (Figure 32.2) means that modelling noise as Gaussian may not be justified.

Large distances between some point pairs could have a significant effect on the estimate of the transformation. The square of a large number is very large indeed, so that reducing a large distance somewhat can justify incurring small to medium error on many other pairs (Figure **??**). A simple procedure to manage this effect is to ignore corresponding pairs if the distance between them is too large. One estimates the transformation using only pairs where distances are small. If

points were omitted in one step of the iteration, they may return in another. This strategy can be helpful, but there is a danger that too many pairs are omitted and the iteration does not converge. Corresponding pairs with large distances between them are likely *outliers* – measurements or data that will not conform to a model, but can have significant impact on estimating the model. Well established procedures for handling outliers are easily adapted to registration problems.

13.3.1 IRLS: Weighting Down Outliers

Rather than just ignoring big distances, one might weight down correspondences that seem implausible. Doing so requires some way to estimate an appropriate set of weights. A large weight for errors at points that are "trustworthy" and a low weight for errors at "suspicious" points should result in a registration that is robust to outliers. We can obtain such weights using a *robust loss*, which will reduce the cost of large errors. This can be seen as modifying the probability model. Gaussian noise tends to produce few large values (which so have very large negative log-likelihood), and we want a model that has higher probability of large errors (equivalently, penalizes them less severely than a normal model would). Write θ for the parameters of the transformation, $\mathcal{T}_\theta$ for the transformation, and $r_i(\mathbf{x}_i, \mathbf{y}_{c(i)}, \theta)$ for the residual error of the model on the ith measurement and its corresponding reference point. For us, r_i will always be $l2norm\mathbf{x}_i - \mathcal{T}_\theta(\mathbf{y}_{c(i)})$. So rather than minimizing

$$\sum_i (r_i(\mathbf{x}_i, \mathbf{y}_{c(i)}, \theta))^2$$

as a function of θ, we will minimize an expression of the form

$$\sum_i \rho(r_i(\mathbf{x}_i, \mathbf{y}_{c(i)}, \theta); \sigma),$$

for some appropriately chosen function ρ. Clearly, our negative log-likelihood is one such estimator (use $\rho(u;\sigma) = u^2$). The trick is to make $\rho(u;\sigma)$ look like u^2 for smaller values of u, but ensure that it grows more slowly than u^2 for larger values of u.

The *Huber loss* uses

$$\rho(u;\sigma) = \begin{cases} \frac{u^2}{2} & |u| < \sigma \\ \sigma|u| - \frac{\sigma^2}{2} & \end{cases}$$

which is the same as u^2 for $-\sigma \leq u \leq \sigma$, switches to $|u|$ for larger (or smaller) σ, and has continuous derivative at the switch. The Huber loss is convex (meaning that there will be a unique minimum for our models) and differentiable, but is not smooth. The choice of the parameter σ (which is known as *scale*) has an effect on the estimate. You should interpret this parameter as the distance that a point can lie from the fitted function while still being seen as an *inlier* (anything that isn't even partially an outlier).

The *Pseudo Huber loss* uses

$$\rho(u;\sigma) = \sigma^2 \left(\sqrt{1 + \left(\frac{u}{\sigma}\right)} - 1 \right).$$

FIGURE 13.5:
TODO: Figure showing a bunch of robust loss functions **TODO:** Source, Credit, Permission

A little fiddling with Taylor series reveals this is approximately u^2 for $|u|/\sigma$ small, and linear for $|u|/\sigma$ big. This has the advantage of being differentiable.

The **** **TODO:** what is this loss called uses

$$\rho(u;\sigma) = \frac{\sigma^2 u^2}{u^2 + \sigma^2}$$

which is approximately u^2 for $|u|$ much smaller than σ, and close to σ^2 for $|u|$ much larger than σ.

Each of these losses increases monotonically in $|u|$ (the absolute value is important here!), so it is always better to reduce the residual. For the Huber loss and the Pseudo-Huber loss, the penalty grows with $|u|$, but grows more slowly with big $|u|$ than with small $|u|$. This implies that the underlying probability model will produce very large distances less often than large distances, but more often than a Gaussian model would. For the **** loss, the penalty eventually increases extremely slowly with increasing $|u|$, implying the underlying probability model is willing to produce arbitrarily large distances on occasion, and that the probability of large distances declines very slowly.

Our minimization criterion is

$$\begin{aligned} \nabla_\theta \left(\sum_i \rho(r(\mathbf{x}_i, \mathbf{y}_i, \theta); \sigma) \right) &= \sum_i \left[\frac{\partial \rho}{\partial u} \right] \nabla_\theta r(\mathbf{x}_i, \mathbf{y}_i, \theta) \\ &= 0. \end{aligned}$$

Here the derivative $\frac{\partial \rho}{\partial u}$ is evaluated at $r(\mathbf{x}_i, \mathbf{y}_i, \theta$, so it is a function of θ. Now notice that

$$\begin{aligned} \sum_i \left[\frac{\partial \rho}{\partial u} \right] \nabla_\theta r(\mathbf{x}_i, \mathbf{y}_i, \theta) &= \sum_i \left[\left(\frac{\frac{\partial \rho}{\partial u}}{r(\mathbf{x}_i, \mathbf{y}_i, \theta)} \right) \right] r(\mathbf{x}_i, \mathbf{y}_i, \theta) \nabla_\theta r(\mathbf{x}_i, \mathbf{y}_i, \theta) \\ &= \sum_i \left[\left(\frac{\frac{\partial \rho}{\partial u}}{r(\mathbf{x}_i, \mathbf{y}_i, \theta)} \right) \right] \nabla_\theta \left[r(\mathbf{x}_i, \mathbf{y}_i, \theta) \right]^2 \\ &= 0. \end{aligned}$$

Now $[r(\mathbf{x}_i, \mathbf{y}_i, \theta)]^2$ is the squared error. If we happened to know the true minimum $\hat{\theta}$ and wrote

$$w_i = \left(\frac{\frac{\partial \rho}{\partial u}}{r(\mathbf{x}_i, \mathbf{y}_i, \theta)} \right)$$

(evaluated at that minimum), then

$$\sum_i w_i \nabla_\theta \left[r(\mathbf{x}_i, \mathbf{y}_i, \theta) \right]^2 = 0$$

at $\theta = \hat{\theta}$. We do not know w_i, but if we did, we already have a recipe to solve this problem for a variety of different transformations (Sections 32.2, 32.2 and 32.2). A natural strategy to adopt is to start with some transformation estimate and unit weights, then repeat:

- **Estimate correspondences** using the estimated transformation. Because all the robust losses are monotonic in $|u|$, finding the closest reference point to each measurement will do.

- **Re-estimate weights** using the new correspondences and the transformation.

- **Re-estimate transformation** using the new correspondences and the new weights, and the closed form algorithms from Sections 32.2, 32.2 and 32.2.

This procedure is known as *iteratively reweighted least squares*

Procedure: 13.4 *Estimating a Transformation from Data with a Robust Loss: Initialization*

Given N known reference points $\mathbf{y}_i = (y_{i,1}, \ldots, y_{i,d})$ in affine coordinates and M measurements $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,d})$, initialize by: **TODO:** What is best? likely translation from cogs, affine / euclidean from second moments, but how do you compute second moments robustly?

Procedure: 13.5 *Estimating a Transformation from Data with a Robust Loss: Iteration*

Start with N known reference points $\mathbf{y}_i = (y_{i,1}, \ldots, y_{i,d})$ in affine coordinates and M measurements $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,d})$, and a transformation estimate $\mathcal{T}_{\theta^{(1)}}$ with parameters θ^1. Form $\mathbf{m}_i^{(1)} = \mathcal{T}_{\theta^{(1)}}(\mathbf{y}_i)$, then iterate:

- for each $\mathbf{x}_i$, find $\mathbf{m}_{c(i)}^{(n+1)}$ that is closest;
- for each pair $(\mathbf{x}_i, \mathbf{m}_{c(i)}^{(n+1)})$, form $u_i = \| \mathbf{x}_i - \mathbf{m}_{c(i)}^{(n+1)} \|_2$ and

$$w_i = \left(\frac{\frac{\partial \rho}{\partial u}}{\mathbf{u}_i} \right);$$

- estimate $\mathcal{T}_{\theta^{(n+1)}}$ using the set of pairs $(\mathbf{x}_i, \mathbf{m}_{c(i)})$ and the weights w_i;
- form $\mathbf{m}_i = \mathcal{T}_{\theta^{(n+1)}}(\mathbf{m}_i)$.

Test for convergence by testing either that the correspondences did not change in a round, or by checking that $\mathcal{T}_{\theta^{(n+1)}}$ is close to the identity. The required transformation is $\mathcal{T}_{\theta^{(n+1)}} \circ \mathcal{T}_{\theta^{(n)}} \circ \ldots \circ \mathcal{T}_{\theta^{(1)}}$.

CHAPTER 14

A Camera Above a Ground Plane

Imagine a camera is moving above a ground plane. Using registration tools together with camera matrices makes means we can calibrate the camera's intrinsics, reason about the position and orientation of the camera, and reconstruct the pattern on the ground plane. In turn, this reconstruction can yield estimates of what objects are moving and whether there are objects that have relief ("stick out" from the ground).

14.1 PIPH: PERPENDICULAR IMAGE PLANE AND FIXED HEIGHT

Assume the camera moves at fixed height above a ground plane, and the ground plane is at right angles to the image plane (call this configuration *PIPH* for short). This is a good model for a camera on (say) an autonomous car or a taxiing aircraft. Figure 14.1 shows the notation, etc. Here the focal length is f, the ground plane is the plane $y = -h$ in the camera coordinate system (remember, z is depth into the scene). Remarkably, we can calibrate the camera with elementary geometric reasoning in a configuration like this, at least for simple cameras.

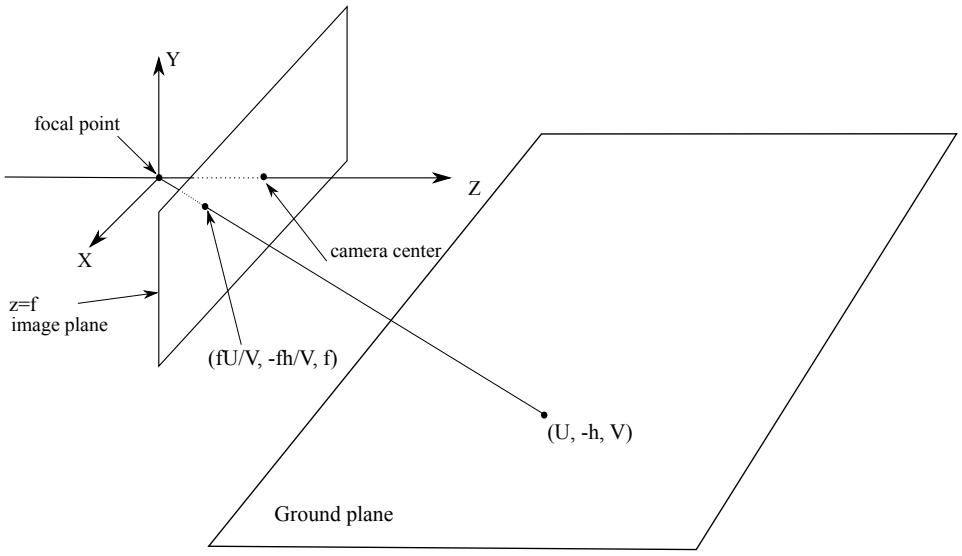

FIGURE 14.1: *A perspective camera with its image plane at right angles to a ground plane ($y = -h$), imaging a point on the ground plane.*

14.1.1 PIPH Geometry

From Figure 14.1, in the camera coordinate system, the point $(u, -h, v)$ on the ground plane in intersects the image plane at $(fu/v, -fh/v, f)$. These are affine coordinates for a point in 3D. Homogeneous coordinates for the point on the image plane are $(u/v, -h/v, 1)$ or equivalently $(u, -h, v)$. Similarly, homogeneous coordinates for the point on the ground plane are $(u, v, 1)$.

To get the transformation from the ground plane in world coordinates to the image in image coordinates, we must account for extrinsics and intrinsics. The homography from the ground plane to the image will be

$$\mathcal{T}_{\mathrm{g}\to\mathrm{i}} = \mathcal{T}_{\mathrm{int}} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -h \\ 0 & 1 & 0 \end{bmatrix} \mathcal{T}_{\mathrm{ext}} = \begin{bmatrix} as & k & c_x \\ 0 & s & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -h \\ 0 & 1 & 0 \end{bmatrix} \mathcal{T}_{\mathrm{ext}}.$$

Here $\mathcal{T}_{\mathrm{int}}$ comes from the camera intrinsics and $\mathcal{T}_{\mathrm{ext}}$ (which represents extrinsics) is a rotation and translation *in the ground plane*. This transformation is present because the coordinate system on the ground plane may not be directly below the focal point and aligned with the camera.

You should check that, in PIPH geometry, the horizon of the ground plane is horizontal in the image and passes through c_y (Figure 14.1 should help), so we can determine c_y from an image. Write (i_x, i_y) for the affine coordinates of a point in the image. If we ensure that the horizon is the line $i_y = 0$ (which we can do by a simple subtraction), then $c_y = 0$. In these coordinates, we have

$$\mathcal{T}_{\mathrm{g}\to\mathrm{i}} = \begin{bmatrix} as & k & c_x \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -h \\ 0 & 1 & 0 \end{bmatrix} \mathcal{T}_{\mathrm{ext}} = \begin{bmatrix} as & c_x & -hk \\ 0 & 0 & -sh \\ 0 & 1 & 0 \end{bmatrix} \mathcal{T}_{\mathrm{ext}}.$$

This is *not* an affine transformation. However

$$\begin{aligned} \mathcal{C}_{\mathrm{g}\to\mathrm{i}}\mathcal{T}_{\mathrm{g}\to\mathrm{i}} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \mathcal{T}_{\mathrm{g}\to\mathrm{i}} \\ &= \begin{bmatrix} as & c_x & -hk \\ 0 & 1 & 0 \\ 0 & 0 & sh \end{bmatrix} \mathcal{T}_{\mathrm{ext}} \\ &\equiv \begin{bmatrix} \frac{a}{h} & \frac{c_x}{sh} & -\frac{k}{s} \\ 0 & \frac{1}{sh} & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathcal{T}_{\mathrm{ext}} \end{aligned}$$

(recall $\equiv$ means that they are the same homography; one is a scaling of the other, which doesn't matter in homogeneous coordinates). This means $\mathcal{C}_{\mathrm{g}\to\mathrm{i}}\mathcal{T}_{\mathrm{g}\to\mathrm{i}}$ is an affine transformation. This is a powerful fact. If we know some points on the ground plane and corresponding points in the image, we can recover $\mathcal{T}_{\mathrm{g}\to\mathrm{i}}$, premultiply by $\mathcal{C}_{\mathrm{g}\to\mathrm{i}}$ (which we know), then read off some camera parameters. Remarkably, we can also estimate camera ground plane motion and the pattern on the ground plane *without* calibrating the camera. These estimates are up to scale – we cannot

get the magnitude of the translation or the size of objects on the ground plane without other information.

14.1.2 PIPH Calibration

Now assume we have a set of points $\mathbf{p}_i$ on the ground plane, and we can find the corresponding points $\mathbf{q}_i$ on the image plane. Fit $\mathcal{T}_{\mathrm{g}\rightarrow\mathrm{i}}$ to this information using procedure 13.2, to obtain $\mathcal{F}$. Now

$$\begin{aligned}\mathcal{C}_{\mathrm{g}\rightarrow\mathrm{i}}\mathcal{F} &\equiv \begin{bmatrix} \frac{a}{h} & \frac{c_x}{sh} & -\frac{k}{s} \\ 0 & \frac{1}{sh} & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathcal{T}_{\text{ext}} \\ &= \begin{bmatrix} \frac{a}{h} & \frac{c_x}{sh} & -\frac{k}{s} \\ 0 & \frac{1}{sh} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathcal{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \\ &= \begin{bmatrix} \mathcal{M} & \mathbf{u} \\ \mathbf{0}^T & 1 \end{bmatrix}\end{aligned}$$

where $\mathcal{T}_{\text{ext}}$ is a Euclidean transformation on the plane. We cannot recover $\frac{k}{s}$ from $\mathbf{u}$ because we don't know $\mathbf{t}$, the translation from the ground plane coordinate system to the camera coordinate system. However, for many cameras $k = 0$, and we assume this is the case for our camera. We have

$$\mathcal{M} = \begin{bmatrix} \frac{a}{h} & -\frac{c_x}{sh} \\ 0 & \frac{1}{sh} \end{bmatrix} \mathcal{R}$$

and we can factor $\mathcal{M}$ using an RQ factorization (see procedure 30.1). Doing so yields $\frac{h}{a}$, c_x and sh.

Notice there is a weakness in this procedure. The homography $\mathcal{T}_{\mathrm{g}\rightarrow\mathrm{i}}$ has a known, special parametric form and we did not impose this form when we estimated the homography. The correct way to resolve this is to minimize the error between $\mathcal{T}_{\mathrm{g}\rightarrow\mathrm{i}}(\mathbf{p}_i)$ and $\mathbf{q}_i$ for a homography of the correct form, using our estimates to provide a start point. This is an example of general recipe for calibration that we shall see again – first, make an estimate of parameters to provide a start point, then polish that estimate using an optimization problem

Procedure: 14.1 *PIPH Calibration: Overview*

Given a set of points $\mathbf{p}_i$ on the ground plane, corresponding points $\mathbf{q}_i$ on the image plane, and a camera known to be in PIPH geometry, estimate camera intrinsics and extrinsics by:

- Assuming $k = 0$;
- Obtaining a start point for $\frac{h}{a}$, c_x, sh and extrinsic parameters as below;
- Polishing the start point by optimization.

Procedure: 14.2 *PIPH Calibration: Initialization*

Initial: Fit $\mathcal{T}_{\mathrm{g}\to\mathrm{i}}$ to the points using procedure 13.2, to obtain $\mathcal{F}$. Now compute

$$\mathcal{C}_{\mathrm{g}\to\mathrm{i}}\mathcal{F} = \begin{bmatrix} \mathcal{M} & \mathbf{u} \\ \mathbf{0} & 1 \end{bmatrix}.$$

Intrinsics start point: Use an RQ factorization on $\mathcal{M}$ to obtain $\mathcal{M} = \mathcal{R}\mathcal{Q}$; $\mathcal{R}$ is upper triangular, and yields $\frac{h}{a}$, c_x, sh. **Extrinsics start point:** We have

$$\mathcal{T}_{\mathrm{ext}} = \begin{bmatrix} \mathcal{Q} & \mathbf{u} \\ \mathbf{0} & 1 \end{bmatrix}$$

Procedure: 14.3 *PIPH Calibration: Optimization*

Solve the optimization problem

$$\sum_i (q_{i,x} - w_{i,x})^2 + (q_{i,y} - w_{i,y})^2$$

where

$$\begin{aligned} w_{i,x} &= \frac{h_{11}p_{i,x} + h_{12}p_{i,y} + h_{13}}{h_{31}p_{i,x} + h_{32}p_{i,y} + h_{33}} \\ w_{i,y} &= \frac{h_{21}p_{i,x} + h_{22}p_{i,y} + h_{23}}{h_{31}p_{i,x} + h_{32}p_{i,y} + h_{33}} \\ \mathcal{H} &= \begin{bmatrix} \frac{a}{h} & \frac{c_x}{sh} & 0 \\ 0 & \frac{1}{sh} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathcal{Q} & \mathbf{u} \\ \mathbf{0}^T & 1 \end{bmatrix} \\ \mathcal{Q} &= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \end{aligned}$$

and the parameters are $\frac{a}{h}$, c_x, sh, θ and $\mathbf{u}$ using the start point of procedure 14.2.

14.1.3 Using PIPH to Estimate Motion

Imagine the camera captures an image $\mathcal{I}_n$ at frame n, moves rigidly, then captures $\mathcal{I}_{n+1}$. The camera image plane stays perpendicular to the ground plane, and the height of the focal point doesn't change. We can recover the camera motion and some camera parameters in this case. We know that $\mathcal{C}_{\mathrm{g}\to\mathrm{i}}\mathcal{T}_{\mathrm{g}\to\mathrm{i}_n}$ and $\mathcal{C}_{\mathrm{g}\to\mathrm{i}}\mathcal{T}_{\mathrm{g}\to\mathrm{i}_{n+1}}$ are both affine. Notice that

$$\mathcal{T}_{\mathrm{i}_n\to\mathrm{i}_{n+1}} = \mathcal{T}_{\mathrm{g}\to\mathrm{i}_{n+1}}\mathcal{T}^{-1}_{\mathrm{g}\to\mathrm{i}_n}.$$

We can *measure* $\mathcal{T}_{\mathrm{i}_n \to \mathrm{i}_{n+1}}$ by finding interest points in the two images, then using Procedure 13.2. Write $\mathcal{F}$ for the measured transformation. We must have that

$$\mathcal{C}_{\mathrm{g}\to\mathrm{i}}\mathcal{F}\mathcal{C}^{-1}_{\mathrm{g}\to\mathrm{i}} = \mathcal{C}_{\mathrm{g}\to\mathrm{i}}\mathcal{T}_{\mathrm{g}\to\mathrm{i}_{n+1}}\mathcal{T}^{-1}_{\mathrm{g}\to\mathrm{i}_n}\mathcal{C}^{-1}_{\mathrm{g}\to\mathrm{i}} = \left[\mathcal{C}_{\mathrm{g}\to\mathrm{i}}\mathcal{T}_{\mathrm{g}\to\mathrm{i}_{n+1}}\right]\left[\mathcal{C}_{\mathrm{g}\to\mathrm{i}}\mathcal{T}_{\mathrm{g}\to\mathrm{i}_n}\right]^{-1}$$

and so $\mathcal{C}_{\mathrm{g}\to\mathrm{i}}\mathcal{F}\mathcal{C}^{-1}_{\mathrm{g}\to\mathrm{i}}$ is affine. In fact, we know the form of this matrix, which is

$$\begin{bmatrix} \frac{a}{h} & \frac{c_x}{sh} & 0 \\ 0 & \frac{1}{sh} & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathcal{T}_{\mathrm{ext}_{n+1}}\mathcal{T}^{-1}_{\mathrm{ext}_n} \begin{bmatrix} \frac{a}{h} & \frac{c_x}{sh} & 0 \\ 0 & \frac{1}{sh} & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1}.$$

Now $\mathcal{E}_{n\to n+1} = \mathcal{T}_{\mathrm{ext}_{n+1}}\mathcal{T}^{-1}_{\mathrm{ext}_n}$ is the camera motion on the ground plane. Notice that

$$\begin{bmatrix} \frac{a}{h} & \frac{c_x}{sh} & 0 \\ 0 & \frac{1}{sh} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & \frac{c_x}{sa} & 0 \\ 0 & \frac{1}{sa} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{a}{h} & 0 & 0 \\ 0 & \frac{a}{h} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and recall that isotropic scaling commutes with rotation (Section 32.2), to find that

$$\mathcal{F} = \begin{bmatrix} 1 & \frac{c_x}{sa} & 0 \\ 0 & \frac{1}{sa} & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathcal{E} \begin{bmatrix} 1 & \frac{c_x}{sa} & 0 \\ 0 & \frac{1}{sa} & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1}.$$

Now write $\mathcal{M} = \mathcal{C}_{\mathrm{g}\to\mathrm{i}}\mathcal{F}\mathcal{C}^{-1}_{\mathrm{g}\to\mathrm{i}}$. The upper 2×2 block of $\mathcal{M}$ is

$$\begin{bmatrix} \cos\theta + \frac{c_x}{sa}\sin\theta & -(sa + \frac{c_x^2}{sa})\sin\theta \\ \frac{1}{sa}\sin\theta & \cos\theta - \frac{c_x}{sa}\sin\theta \end{bmatrix}$$

so we can recover the rotation from

$$\cos\theta = m_{11} + m_{22},$$

and some calibration parameters from

$$\begin{aligned} s^2a^2 &= \frac{1-\cos^2\theta}{m_{12}^2} \\ \left(\frac{c_x}{as}\right)^2 &= \frac{(m_{22}-\cos\theta)^2}{(1-\cos^2\theta)}. \end{aligned}$$

This means we can recover as and c_x if we can determine the signs of the square roots. But a and s are necessarily positive and we can obtain the sign of c_x by elementary reasoning about the camera, so we can recover the signs. Now if the camera translates by $[t_x, t_y]$, then the translation component of $\mathcal{M}$ is

$$\begin{bmatrix} m_{13} \\ m_{23} \end{bmatrix} = \begin{bmatrix} -\frac{h}{a}t_x + \frac{hc_x}{a}t_y \\ -sht_y \end{bmatrix}$$

so that

$$\frac{-(as)m_{13} + \frac{c_x}{as}m_{23}}{-m_{23}} = \frac{t_x}{t_y}.$$

Now we *observe* u and v, and can recover as and c_x, so we know the *direction* but not magnitude of the translation. Equivalently, we can recover the movement of the camera up to scale.

Procedure: 14.4 *PIPH Motion Estimation*

Given an image $\mathcal{I}_n$ at frame n and $\mathcal{I}_{n+1}$ at frame $n+1$, where an uncalibrated camera with $k = 0$ moves rigidly, with image plane perpendicular to the ground plane, and the height of the focal point fixed but unknown, obtain an estimate of $\mathcal{T}_{\mathcal{I}_n \to \mathcal{I}_{n+1}}$; write $\mathcal{F}$ for this estimate. Find this by identifying interest points in $\mathcal{I}_n$ and $\mathcal{I}_{n+1}$, and fitting a homography to these interest points (Procedure 13.2). Then $\mathcal{M} = \mathcal{C}_{\mathrm{g}\to\mathrm{i}} \mathcal{F} \mathcal{C}_{\mathrm{g}\to\mathrm{i}}^{-1}$ is affine. We have

$$\begin{aligned} \cos\theta &= m_{11} + m_{22} \\ s^2a^2 &= \frac{1-\cos^2\theta}{m_{12}^2} \\ \left(\frac{c_x}{as}\right)^2 &= \frac{(m_{22}-\cos\theta)^2}{(1-\cos^2\theta)} \end{aligned}$$

yielding rotation and some camera parameters. The translation is recovered from

$$\frac{-(as)m_{13} + \frac{c_x}{as}m_{23}}{-m_{23}} = \frac{t_x}{t_y}.$$

14.1.4 The Pattern on the Ground Plane

We can recover the pattern on the ground plane up to scale as well from two images. Write the true pattern $\mathcal{P}$. Recall that $\mathcal{C}_{\mathrm{g}\to\mathrm{i}}\mathcal{T}_{\mathrm{g}\to\mathrm{i}}$ is affine, which means that $\mathcal{T}_{\mathrm{i}\to\mathrm{g}}\mathcal{C}_{\mathrm{g}\to\mathrm{i}}^{-1}$ is affine as well (Section 32.2). This means that if we apply the homography $\mathcal{C}_{\mathrm{g}\to\mathrm{i}}$ to the image, we will obtain a pattern that is within an affine transformation of the ground plane, and we can determine the form of the affine transformation. This is easy to do, because $\mathcal{C}_{\mathrm{g}\to\mathrm{i}}$ is known.

We have

$$\mathcal{T}_{\mathrm{g}\to\mathrm{i}} = \begin{bmatrix} as & c_x & -hk \\ 0 & 0 & -sh \\ 0 & 1 & 0 \end{bmatrix} \mathcal{T}_{\mathrm{ext}}.$$

so that

$$\mathcal{T}_{\mathrm{i}\to\mathrm{g}}\mathcal{C}_{\mathrm{g}\to\mathrm{i}}^{-1} = \mathcal{T}_{\mathrm{ext}}^{-1} \begin{bmatrix} as & c_x & -hk \\ 0 & 0 & -sh \\ 0 & 1 & 0 \end{bmatrix}^{-1} \mathcal{C}_{\mathrm{g}\to\mathrm{i}}^{-1}.$$

Now as in Section 14.1.2, we assume $k = 0$. We have

$$\mathcal{C}_{\mathrm{g}\to\mathrm{i}} \begin{bmatrix} as & c_x & 0 \\ 0 & 0 & -sh \\ 0 & 1 & 0 \end{bmatrix} \equiv \begin{bmatrix} \frac{a}{h} & \frac{c_x}{sh} & 0 \\ 0 & \frac{1}{sh} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

so that

$$\mathcal{T}_{\mathrm{i}\to\mathrm{g}}\mathcal{C}^{-1}_{\mathrm{g}\to\mathrm{i}} = \mathcal{T}^{-1}_{\mathrm{ext}} \begin{bmatrix} \frac{h}{a} & 0 & 0 \\ 0 & \frac{h}{a} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -c_x & 0 \\ 0 & as & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

In turn, if we know c_x and as, we can recover the image plane pattern up to scale. As Section 14.1.3 shows, these parameters can be estimated from two distinct views of the ground plane.

Procedure: 14.5 *PIPH Pattern Estimation*

Given an image $\mathcal{I}_n$ at frame n and $\mathcal{I}_{n+1}$ at frame $n+1$, where an uncalibrated camera with $k = 0$ moves rigidly, with image plane perpendicular to the ground plane, and the height of the focal point fixed but unknown, obtain camera parameters as and c_x from procedure 14.4. Write

$$\mathcal{T}_{\mathrm{partial}} = \begin{bmatrix} 1 & -c_x & 0 \\ 0 & as & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Then the ground plane pattern is within a scale of

$$\mathcal{T}^{-1}_{\mathrm{partial}}\mathcal{C}_{\mathrm{g}\to\mathrm{i}}(\mathcal{I}_n)$$

14.1.5 Off Perpendicular Image Planes

All the procedures above can be extended to deal with an off-perpendicular image plane if one is allowed a single calibration step. This step essentially estimates the angle between the image plane and the ground plane. In particular, notice that the methods of Section 14.1.3 and 14.5 depend on the fact that a *known* homography applied to the image yields something that is within an affine transformation of the ground plane.

When the image plane is not perpendicular to the ground plane, the homography from ground plane to image can be derived from Figure 32.2 (assuming $k = 0$) as

$$\mathcal{P}_{\mathrm{g}\to\mathrm{i}} = \begin{bmatrix} as & 0 & c_x \\ 0 & s & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \gamma & -h \\ 0 & 1 & 0 \end{bmatrix} \mathcal{T}_{\mathrm{ext}} = \begin{bmatrix} as & c_x & -hk \\ 0 & s\gamma + c_y & -hs \\ 0 & 1 & 0 \end{bmatrix} \mathcal{T}_{\mathrm{ext}}.$$

You should check that

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & \frac{1}{s\gamma + c_y} & -1 \end{bmatrix} \mathcal{P}_{\mathrm{g}\to\mathrm{i}} \equiv \begin{bmatrix} \frac{a}{h} & \frac{c_x}{hs} & 0 \\ 0 & \frac{1}{hs} & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathcal{T}_{\mathrm{ext}}.$$

This means that, if we can estimate $s\gamma + c_y$, we can apply the strategies of the previous section. A simple strategy for doing so is to image a set of reference points on the ground plane, compute the homography from ground plane to image $\mathcal{P}_{\mathrm{g}\to\mathrm{i}}$, then obtain

$$\mathcal{D}_w = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & w & -1 \end{bmatrix}$$

such that $\mathcal{D}_w\mathcal{P}_{\mathrm{g}\to\mathrm{i}}$ is affine.

14.1.6 PIPH Mosaics

14.2 CAMERA CALIBRATION FROM PLANE REFERENCES

It is possible to calibrate a camera with a plane calibration object, but you need to have several views. Plane patterns are easy to make and easy to disseminate. Obtain a plane pattern with a set of easily localized points (a checkerboard is good) where the locations of those points on the plane are known in world units. So if one is using a checkerboard, one might know that the checks are square and 10cm on edge, for example. Lay this down flat, and take a set of images of it from different views. In each view the calibration points should be visible.

For each view, we will compute the homography from the calibration object's plane to the image plane using point correspondences (Section 32.2). It turns out that these homographies yield constraints on the camera matrix (Section 32.2) and these constraints yield a camera estimate (Section 32.2). This estimate is a start point for an optimization problem (Section 32.2, very much on the lines of Section 32.2).

14.2.1 Constraining Intrinsics with Homographies

Each map from the pattern to an image is a homography. Choose the world coordinate system so that the pattern lies on the plane $z = 0$. Doing so just changes the camera *extrinsics*, so no generality has been lost, but it allows us to write the homography in a useful form. Recall the camera is

$$\mathcal{T}_i\mathcal{C}_p\mathcal{T}_{e,j}$$

where $\mathcal{T}_{e,j}$ is the euclidean transformation giving the extrinsics for the j'th view. This is applied to a set of points $(s_{x,i}, s_{y,i}, 0, 1)$. In turn, the homography for the j'th view must have the form

$$\lambda_j\mathcal{M}_j = \mathcal{T}_i\left[\mathbf{r}_{1,j}, \mathbf{r}_{2,j}, \mathbf{t}_j\right]$$

(where $\mathbf{r}_{1,j}$, $\mathbf{r}_{2,j}$ are the first two *columns* of the rotation matrix in $\mathcal{T}_{e,j}$ and $\mathbf{t}_j$ is the translation). We do not know λ_j (which is non-zero) because scaling the homography matrix yields the same homography. Now write $\mathcal{N}_j = \mathcal{T}_i^{(-1)}\mathcal{M}_j = [\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3]$. We must have that

$$\mathbf{n}_1^T\mathbf{n}_1 - \mathbf{n}_2^T\mathbf{n}_2 = 0 \text{ and } \mathbf{n}_1^T\mathbf{n}_2 = 0.$$

These equations constrain the unknown values of $\mathcal{T}_i$, and we get two for each homography. In turn, with sufficient views (and so homographies), we can estimate $\mathcal{T}_i$.

14.2.2 Estimating Intrinsics from Homographies

In the j'th view of the plane calibration object, we recover a homography $\mathcal{M}_j$. Now write $\mathcal{N}_j = \mathcal{T}_i^{(-1)}\mathcal{M}_j = [\mathbf{n}_{j,1}, \mathbf{n}_{j,2}, \mathbf{n}_{j,3}]$. We know from Section 32.2 that

$$\mathbf{n}_{j,1}^T\mathbf{n}_{j,1} - \mathbf{n}_{j,2}^T\mathbf{n}_{j,2} = 0 \text{ and } \mathbf{n}_{j,1}^T\mathbf{n}_{j,2} = 0.$$

Now write $\mathcal{A} = (\mathcal{T}_i^{(-T)}\mathcal{T}_i^{(-1)})$ (which is unknown). These two constraints are linear homogenous equations in the entries of $\mathcal{A}$, which is 3×3 but symmetric, and so has 6 unknown parameters. If we have 3 homographies, we will have 6 constraints, and can use least squares to recover a 1D family of solutions $\lambda\mathcal{B}$, where $\mathcal{B}$ is *known* and λ is a scale. We now need to find $\mathcal{T}_i$ and λ so that $\lambda\mathcal{B}$ is close to $(\mathcal{T}_i^{(-T)}\mathcal{T}_i^{(-1)}))$.

There are constraints here. Write $\mathcal{U} = \mathcal{T}_i^{(-1)}$. Recall $\mathcal{T}_i$ is upper triangular, and $i_{33} = 1$. This means that $\mathcal{U}$ is upper triangular, and $u_{33} = 1$. We will find $\mathcal{U}$ and λ by finding $\mathcal{V}$ such that $\mathcal{V}^T\mathcal{V}$ is closest to $\mathcal{B}$, then computing $\mathcal{U} = (1/v_{33})\mathcal{V}$.

Finding $\mathcal{V}$ is straightforward. We obtain the closest symmetric matrix to $\mathcal{B}$, then apply a Cholesky factorization (Section 32.2). The factorization could be modified if a very small number appears on the diagonal, but this event is most unlikely. We now invert $\mathcal{U}$ to obtain an estimate $\mathcal{E}$ of $\mathcal{T}_i$. Recall this has the form

$$\begin{bmatrix} as' & k' & c_x' \\ 0 & s' & c_y' \\ 0 & 0 & 1 \end{bmatrix}.$$

so we have $c_x' = e_{13}$, $c_y' = e_{23}$, $s' = e_{22}$, $a = e_{11}/e_{22}$ and $k' = e_{12}$. This is usually an acceptable start point for optimization.

14.2.3 Estimating Extrinsics from Homographies

We have an estimate of the camera intrinsics, and now need an estimate of the extrinsics for each view. Recall from Section **??** that

$$\lambda_j\mathcal{M}_j = \mathcal{T}_i\left[\mathbf{r}_{1,j}, \mathbf{r}_{2,j}, \mathbf{t}_j\right]$$

(where $\mathbf{r}_{1,j}$, $\mathbf{r}_{2,j}$ are the first two *columns* of the rotation matrix in $\mathcal{T}_{e,j}$ and $\mathbf{t}_j$ is the translation). We have estimates of $\mathcal{M}_j$ and of $\mathcal{T}_i$, but we do not know λ_j. We can solve for λ_j by noticing that the first two columns of

$$\lambda_j\mathcal{T}_i^{-1}\mathcal{M}_j = \lambda_j\mathcal{Q}_j = \lambda_j\left[\mathbf{q}_{1,j}, \mathbf{q}_{2,j}, \mathbf{q}_{3,j}\right]$$

are unit vectors, and are normal to one another. For example, we might estimate

$$\lambda_j = \sqrt{\frac{2}{\mathbf{q}_{1,j}^T\mathbf{q}_{1,j} + \mathbf{q}_{2,j}^T\mathbf{q}_{2,j}}}$$

and from this follows the estimate

$$\mathcal{T}_{e,j} = \begin{bmatrix} \lambda_j\mathbf{q}_{1,j} & \lambda_j\mathbf{q}_{2,j} & \lambda_j^2\mathbf{q}_{1,j} \times \mathbf{q}_{2,j} & \lambda_j\mathbf{q}_{3,j} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

14.2.4 Formulating the Optimization Problem

As in Section 32.2, we will calibrate the camera by solving an optimization problem. The optimization problem is relatively straightforward to formulate, and follows the same lines as that in Section 32.2. The main difference with that section is that all calibration points will lie on the plane $z = 0$ in world coordinates, and we will have more than one view of that plane. Write $\mathbf{t}_{ij} = [t_{x,ij}, t_{y,ij}]$ for the measured x, y position in the image plane of the i'th reference point in the j'th view. We have that $\mathbf{t}_{ij} = \hat{\mathbf{t}}_{ij} + \xi_{ij}$, where ξ_{ij} is an error vector and $\hat{\mathbf{t}}_{ij}$ is the true (unknown) position. Again, assume the error is isotropic, so it is natural to minimize

$$\sum_{ij} \xi_{ij}^T \xi_{ij}.$$

The main issue here is writing out expressions for ξ_{ij} in the appropriate coordinates. Write $\mathcal{T}_i$ for the intrinsic matrix whose u, v'th component will be i_{uv}; $\mathcal{T}_{e,j}$ for the j'th extrinsic transformation, whose u, v'th component will be e_{uv}; and $\mathbf{s}_i = [s_{x,i}, s_{y,i}, 0]$ for the known coordinates of the i'th reference point in the coordinate frame of the reference points. Recalling that $\mathcal{T}_i$ is lower triangular, and engaging in some manipulation, we obtain

$$\sum_{ij} \xi_{ij}^T \xi_{ij} = \sum_{i} (t_{x,ij} - p_{x,ij})^2 + (t_{y,ij} - p_{y,ij})^2$$

where

$$\begin{aligned} p_{x,ij} &= \frac{i_{11} g_{x,ij} + i_{12} g_{y,ij} + i_{13} g_{z,ij}}{g_{z,ij}} \\ p_{y,ij} &= \frac{i_{22} g_{x,ij} + i_{23} g_{z,ij}}{g_{z,ij}} \end{aligned}$$

and

$$\begin{aligned} g_{x,ij} &= e_{11,j} s_{x,i} + e_{12,j} s_{y,i} + e_{14,j} \\ g_{y,ij} &= e_{21,j} s_{x,i} + e_{22,j} s_{y,i} + e_{24,j} \\ g_{z,ij} &= e_{31,j} s_{x,i} + e_{32,j} s_{y,i} + e_{34,j} \end{aligned}$$

(which you should check as an exercise – notice the missing $s_{z,i}$ terms!). This is a constrained optimization problem, because $\mathcal{T}_e$ is a Euclidean transformation. The constraints here are

$$\begin{aligned} 1 - \sum_v e_{j,1v}^2 = 0 \text{ and } 1 - \sum_v e_{j,2v}^2 = 0 \text{ and } 1 - \sum_v e_{j,3v}^2 = 0 \\ \sum_v e_{j,1v} e_{j,2v} = 0 \text{ and } \sum_v e_{j,1v} e_{j,3v} = 0 \text{ and } 1 - \sum_v e_{j,2v} e_{j,3v} = 0 \end{aligned} .$$

As in Section 32.2, we could just throw this into a constrained optimizer (review Section 32.2), but good behavior requires a good start point.

Procedure: 14.6 *Calibrating a Camera from Multiple Homographies*

Procedure: 14.7 *Calibrating a Camera from Multiple Homographies: Start Point*

CHAPTER 15

Using Camera Models

Registration tools make it possible to reason about position and orientation of manouvering cameras, movement of a camera at fixed height, and to calibrate a camera in two ways.

15.1 CAMERA CALIBRATION FROM A 3D REFERENCE

Camera calibration involves estimating the intrinsic parameters of the camera, and perhaps lens parameters if needed, from one or more images. There are numerous strategies, all using the following recipe: build a *calibration object*, where the positions of some points (*calibration points*) are known; view that object from one or more viewpoints; obtain the image locations of the calibration points; and solve an optimization problem to recover camera intrinsics and perhaps lens parameters. As one would expect, much depends on the choice of calibration object. If all the calibration points sit on an object, the extrinsics will yield the *pose* (for position and orientation) of the object with respect to the camera. We use a two step procedure: formulate the optimization problem, then find a good starting point.

15.1.1 Formulating the Optimization Problem

The optimization problem is relatively straightforward to formulate. Notation is the main issue. We have N reference points $\mathbf{s}_i = [s_{x,i}, s_{y,i}, s_{z,i}]$ with known position in some reference coordinate system in 3D. The measured location in the image for the i'th such point is $\hat{\mathbf{t}}_i = \left[\hat{t}_{x,i}, \hat{t}_{y,i}\right]$. There may be measurement errors, so the $\hat{\mathbf{t}}_i = \mathbf{t}_i + \xi_i$, where ξ_i is an error vector and $\mathbf{t}$ is the unknown true position. We will assume the magnitude of error does not depend on direction in the image plane (it is *isotropic*), so it is natural to minimize the squared magnitude of the error

$$\sum_i \xi_i^T \xi_i.$$

The main issue here is writing out expressions for ξ_i in the appropriate coordinates. Write $\mathcal{T}_i$ for the intrinsic matrix whose u, v'th component will be i_{uv}; $\mathcal{T}_e$ for the extrinsic transformation, whose u, v'th component will be e_{uv}. Recalling that $\mathcal{T}_i$ is lower triangular, and engaging in some manipulation, we obtain

$$\sum_i \xi_i^T \xi_i = \sum_i (t_{x,i} - p_{x,i})^2 + (t_{y,i} - p_{y,i})^2$$

where

$$\begin{aligned} p_{x,i} &= \frac{i_{11} g_{x,i} + i_{12} g_{y,i} + i_{13} g_{i,3}}{g_{i,3}} \\ p_{y,i} &= \frac{i_{22} g_{x,i} + i_{23} g_{i,3}}{g_{i,3}} \end{aligned}$$

and

$$\begin{aligned} g_{x,i} &= e_{11}s_{x,i} + e_{12}s_{y,i} + e_{13}s_{z,i} + e_{14} \\ g_{y,i} &= e_{21}s_{x,i} + e_{22}s_{y,i} + e_{23}s_{z,i} + e_{24} \\ g_{z,i} &= e_{31}s_{x,i} + e_{32}s_{y,i} + e_{33}s_{z,i} + e_{34} \end{aligned}$$

(which you should check as an exercise). This is a constrained optimization problem, because $\mathcal{T}_e$ is a Euclidean transformation. The constraints here are

$$1 - \sum_v e_{j,1v}^2 = 0 \text{ and } 1 - \sum_v e_{j,2v}^2 = 0 \text{ and } 1 - \sum_v e_{j,3v}^2 = 0$$
$$\sum_v e_{j,1v}e_{j,2v} = 0 \text{ and } \sum_v e_{j,1v}e_{j,3v} = 0 \text{ and } 1 - \sum_v e_{j,2v}e_{j,3v} = 0 \quad .$$

We might just throw this into a constrained optimizer (review Section 32.2), but good behavior requires a good start point. This can be obtained by a little manipulation, which I work through in the next section. Some readers may prefer to skip this at first (or even higher) reading because it's somewhat specialized, but it shows how the practical application of some tricks worth knowing.

15.1.2 Setting up a Start Point

Write $\mathbf{C}_j^T$ for the j'th row of the camera matrix, and $\mathbf{S}_i = [s_{x,i}, s_{y,i}, s_{z,i}, 1]^T$ for homogeneous coordinates representing the i'th point in 3D. Then, assuming no errors in measurement, we have

$$\hat{t}_{x,i} = \frac{\mathbf{C}_1^T\mathbf{S}_i}{\mathbf{C}_3^T\mathbf{S}_i} \text{ and } \hat{t}_{y,i} = \frac{\mathbf{C}_2^T\mathbf{S}_i}{\mathbf{C}_3^T\mathbf{S}_i},$$

which we can rewrite as

$$\mathbf{C}_3^T\mathbf{S}_i\hat{t}_{x,i} - \mathbf{C}_1^T\mathbf{S}_i = 0 \text{ and } \mathbf{C}_3^T\mathbf{S}_i\hat{t}_{y,i} - \mathbf{C}_2^T\mathbf{S}_i = 0.$$

We now have two homogenous linear equations in the camera matrix elements for each pair (3D point, image point). There are a total of 12 degrees of freedom in the camera matrix, meaning we can recover a least squares solution from six point pairs. The solution will have the form $\lambda\mathcal{P}$ where λ is an unknown scale and $\mathcal{P}$ is a known matrix. This is a natural consequence of working with homogeneous equations, but also a natural consequence of working with homogeneous coordinates. You should check that if $\mathcal{P}$ is a projection from projective 3D to the projective plane, $\lambda\mathcal{P}$ will yield the same projection as long as $\lambda \neq 0$.

This is enough information to recover the focal point of the camera. Recall that the focal point is the single point that images to $[0, 0, 0]^T$. This means that if we are presented with a 3×4 matrix claiming to be a camera matrix, we can determine what the focal point of that camera is without fuss – just find the null space of the matrix. Notice that we do not need to know λ to estimate the null space.

Remember this: *Given a 3×4 camera matrix $\mathcal{P}$, the homogeneous coordinates of the focal point of that camera are given by $\mathbf{X}$, where $\mathcal{P}\mathbf{X} = [0, 0, 0]^T$*

There is an important relationship between the focal point of the camera and the extrinsics. Assume that, in the world coordinate system, the focal point can be represented by $\left[\mathbf{f}^T, 1\right]^T$. This point must be mapped to $[0, 0, 0, 1]^T$ by $\mathcal{T}_e$. Because we can recover $\mathbf{f}$ from $\mathcal{P}$ easily, we have an important constraint on $\mathcal{T}_e$, given in the box.

Remember this: *Assume camera matrix $\mathcal{P}$ has null space $\lambda\mathbf{u} = \lambda\left[\mathbf{f}^T, 1\right]^T$. Then we must have $\mathcal{T}_e\mathbf{u} = [0, 0, 0, 1]^T$, so we must have*

$$\mathcal{T}_e = \begin{bmatrix} \mathcal{R} & -\mathcal{R}\mathbf{f} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

This means that, if we know $\mathcal{R}$, we can recover the translation from the focal point. We must now recover the intrinsic transformation and $\mathcal{R}$ from what we know.

$$\lambda\mathcal{P} = \mathcal{T}_i \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathcal{R} & -\mathcal{R}\mathbf{f} \\ \mathbf{0}^T & 1 \end{bmatrix} = \begin{bmatrix} \mathcal{T}_i\mathcal{R} & -\mathcal{T}_i\mathcal{R}\mathbf{f} \end{bmatrix}$$

We do not know λ, but we do know $\mathcal{P}$. Now write $\mathcal{P}_l$ for the left 3×3 block of $\mathcal{P}$, and recall that $\mathcal{T}_i$ is upper triangular and $\mathcal{R}$ orthonormal. The first question is the sign of λ. We expect $\mathsf{Det}(\mathcal{R}) = 1$, and $\mathsf{Det}(\mathcal{T}_i) > 0$, so $\mathsf{Det}(\mathcal{P}_l)$ should be positive. This yields the sign of λ – choose a sign $s \in \{-1, 1\}$ so that $\mathsf{Det}(s\mathcal{P}_l)$ is positive.

We can now factor $s\mathcal{P}_l$ into an upper triangular matrix $\mathcal{T}$ and an orthonormal matrix $\mathcal{Q}$. This is an RQ factorization (Section 32.2). Recall we could not distinguish between scaling caused by the focal length and scaling caused by pixel scale, so that

$$\mathcal{T}_i = \begin{bmatrix} as & k & c_x \\ 0 & s & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

In turn, we have $\lambda = s(1/t_{33})$, $c_y = (t_{23}/t_{33})$, $s = (t_{22}/t_{33})$, $c_x = (t_{13}/t_{33})$, $k = (t_{12}/t_{33})$, and $a = (t_{11}/t_{22})$.

Procedure: 15.1 *Calibrating a Camera using 3D Reference Points*

For N reference points $\mathbf{s}_i = [s_{x,i}, s_{y,i}, s_{z,i}]$ with known position in some reference coordinate system in 3D, write the measured location in the image for the i'th such point $\hat{\mathbf{t}}_i = \left[\hat{t}_{x,i}, \hat{t}_{y,i}\right]$. Now minimize

$$\sum_i \xi_i^T \xi_i = \sum_i (\hat{t}_{x,i} - p_{x,i})^2 + (\hat{t}_{y,i} - p_{y,i})^2$$

where

$$\begin{aligned} p_{x,i} &= \frac{i_{11}g_{x,i} + i_{12}g_{y,i} + i_{13}g_{i,3}}{g_{i,3}} \\ p_{y,i} &= \frac{i_{22}g_{x,i} + i_{23}g_{i,3}}{g_{i,3}} \end{aligned}$$

and

$$\begin{aligned} g_{x,i} &= e_{11}s_{x,i} + e_{12}s_{y,i} + e_{13}s_{z,i} + e_{14} \\ g_{y,i} &= e_{21}s_{x,i} + e_{22}s_{y,i} + e_{23}s_{z,i} + e_{24} \\ g_{z,i} &= e_{31}s_{x,i} + e_{32}s_{y,i} + e_{33}s_{z,i} + e_{34} \end{aligned}$$

subject to:

$$1 - \sum_v e_{j,1v}^2 = 0 \text{ and } 1 - \sum_v e_{j,2v}^2 = 0 \text{ and } 1 - \sum_v e_{j,3v}^2 = 0$$

$$\sum_v e_{j,1v}e_{j,2v} = 0 \text{ and } \sum_v e_{j,1v}e_{j,3v} = 0 \text{ and } 1 - \sum_v e_{j,2v}e_{j,3v} = 0 \quad .$$

Use the start point of procedure 15.2

Procedure: 15.2 *Calibrating a Camera using 3D Reference Points: Start Point*

Estimate the rows of the camera matrix $\mathbf{C}_i$ using at least six points and

$$\mathbf{C}_3^T \mathbf{S}_i \hat{t}_{x,i} - \mathbf{C}_1^T \mathbf{S}_i = 0 \text{ and } \mathbf{C}_3^T \mathbf{S}_i \hat{t}_{y,i} - \mathbf{C}_2^T \mathbf{S}_i = 0.$$

Write $\lambda\mathcal{P}$ for the 1D family of solutions to this set of homogeneous linear equations, organized into 3×4 matrix form. Compute the vector $\mathbf{n} = \left[\mathbf{f}^T, 1\right]$ such that $\mathcal{P}\mathbf{n}$. Write $\mathcal{P}_l$ for the left 3×3 block of $\mathcal{P}$. Choose $s \in \{-1, 1\}$ such that $\mathsf{Det}\left(s\mathcal{P}_l\right) > 0$. Use RQ factorization to obtain $\mathcal{T}$ and $\mathcal{Q}$ such that $s\mathcal{P}_l = \mathcal{T}\mathcal{Q}$. Then the start point for the intrinsic parameters is:

$$\begin{bmatrix} a \\ s \\ k \\ c_x \\ c_y \end{bmatrix} = \begin{bmatrix} (t_{11}/t_{22}) \\ (t_{22}/t_{33}) \\ (t_{12}/t_{33}) \\ (t_{13}/t_{33}) \\ (t_{23}/t_{33}) \end{bmatrix}$$

and for $\mathcal{T}_e$ is:

$$\begin{bmatrix} \mathcal{Q} & -\mathcal{Q}\mathbf{f} \\ \mathbf{0} & 1 \end{bmatrix}.$$

15.2 CALIBRATING THE EFFECTS OF LENS DISTORTION

Now assume the lens applies some form of geometric distortion, as in Section 32.2. There are now strong standard models of the major lens distortions (Section 32.2). We will now estimate lens parameters, camera intrinsics and camera extrinsics from a view of a calibration object (as in Section 32.2; note the methods of Section 32.2 apply to this problem too). As in those sections, we use a two step procedure: formulate the optimization problem (Section 32.2), then find a good starting point (Section 32.2).

15.2.1 Modelling Geometric Lens Distortion

Geometric distortions caused by lenses are relatively easily modelled by assuming the lens causes (x, y) in the image plane to map to $(x+\delta x, y+\delta y)$ in the image plane. We seek a model for $\delta x, \delta y$ that has few parameters and that captures the main effects. A natural model of barrel distortion is that points are "pulled" toward the camera center, with points that are further from the center being "pulled" more. Similarly, pincushion distortion results from points being "pushed" away from the camera center, with distant points being pushed further (Figure **??**).

Set up a polar coordinate system (r, θ) in the image plane using the image center as the origin. The figure and description suggest that barrel and pincushion distortion can be described by a map $(r, \theta) \rightarrow (r + \delta r, \theta)$. We model δr as a polynomial in r. Brown and Conrady [] established the model $\delta r = k_1 r^3 + k_2 r^5$ as sufficient for a wide range of distortions, and we use $(r, \theta) \rightarrow (r + k_1 r^3 + k_2 r^5, \theta)$

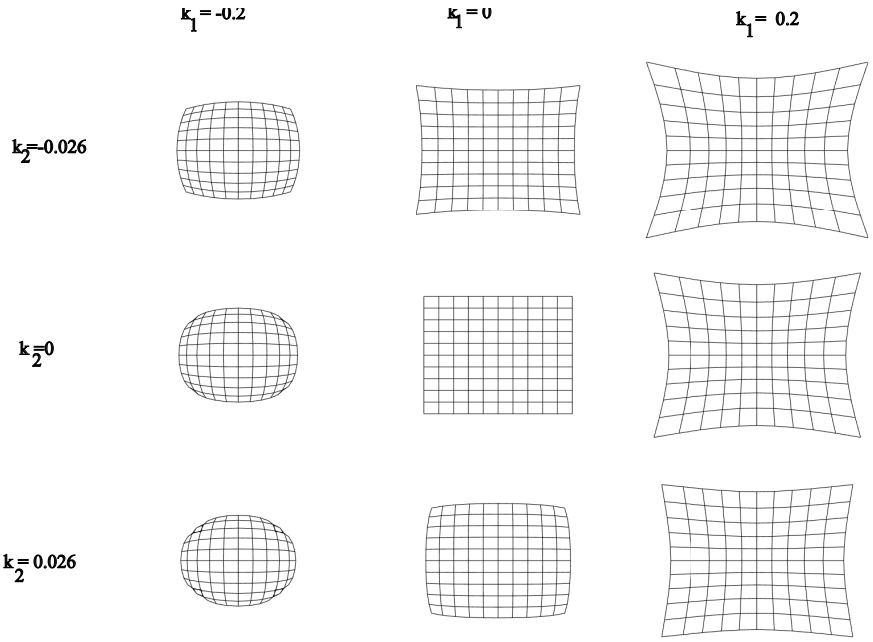

FIGURE 15.1: *The effects of k_1 and k_2 on a neutral grid (***center***), showing how the parameters implement various barrel or pincushion distortions. Notice how k_2 slightly changes the shape of the curves that k_1 produces from straight lines in the grid.*

for unknown k_1, k_2. We must map this model to image coordinates to obtain a map $(x, y) \rightarrow (x + \delta x, y + \delta y)$. Since $\cos\theta = x/r$, $\sin\theta = y/r$, we have $(x, y) \rightarrow (x + x(k_1(x^2 + y^2) + k_2(x^2 + y^2)^2), y + y(k_1(x^2 + y^2) + k_2(x^2 + y^2)^2))$. Figure 15.1 shows distortions resulting from different choices of k_1 and k_2. This model is known as a *radial distortion model*.

More sophisticated lens distortion models account for the lens being off-center. This causes *tangential distortion* (Figure 15.2). The most commonly used model of tangential distortion is a map $(x, y) \rightarrow (x + p_1(x^2 + y^2 + 2x^2) + 2p_2xy, y + p_2(x^2 + y^2 + 2y^2) + 2p_1xy)$ (derived from []; more detail in, for example []).

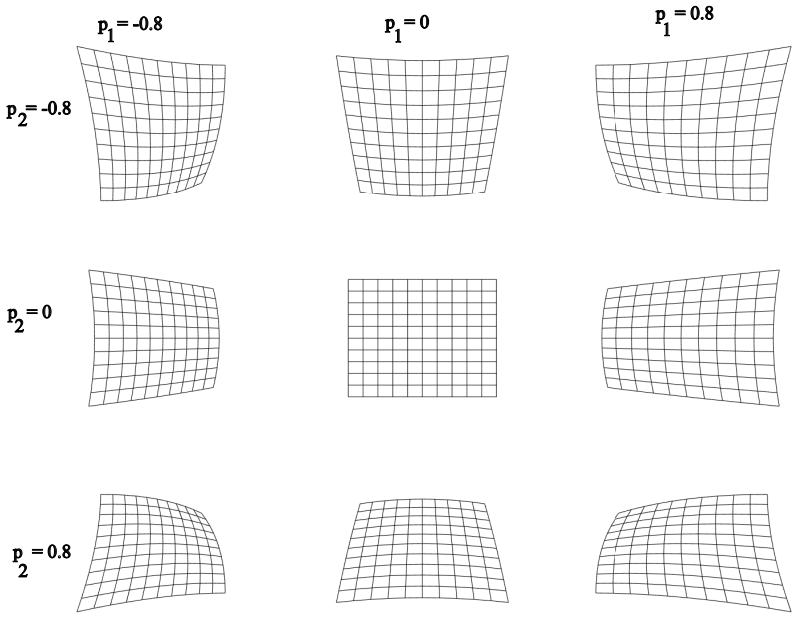

FIGURE 15.2: *The effects of p_1 and p_2 on a neutral grid (***center***), showing how the parameters implement various distortions. These parameters model effects that occur because the lens is off-center; note the grid "turning away" from the lens.*

Remember this: *A full lens distortion model is*

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x + x(k_1(x^2+y^2) + k_2(x^2+y^2)^2) + p_1(x^2+y^2+2x^2) + 2p_2xy \\ y + y(k_1(x^2+y^2) + k_2(x^2+y^2)^2) + p_2(x^2+y^2+2y^2) + 2p_1xy \end{pmatrix}.$$

for k_1, k_2, p_1, p_2 parameters. It is common to ignore tangential distortion and focus on radial distortion by setting $p_1 = p_2 = 0$.

15.2.2 Formulating the Optimization Problem

The optimization problem is relatively straightforward to formulate. Notation is the main issue. Write $\mathbf{t}_i = [t_{x,i}, t_{y,i}]$ for the measured x, y position in the image plane of the i'th reference point. We have that $\mathbf{t}_i = \hat{\mathbf{t}}_i + \xi_i$, where ξ_i is an error vector and $\hat{\mathbf{t}}$ is the true (unknown) position. Again, assume the error is isotropic,

so it is natural to minimize

$$\sum_i \xi_i^T \xi_i.$$

The main issue here is writing out expressions for $\xi_{i,j}$ in the appropriate coordinates. Write $\mathcal{T}_i$ for the intrinsic matrix whose u, v'th component will be i_{uv}; $\mathcal{T}_e$ for the extrinsic transformation, whose u, v'th component will be e_{uv}; and $\mathbf{s}_i = [s_{x,i}, s_{y,i}, s_{z,i}]$ for the known coordinates of the i'th reference point in the coordinate frame of the reference points. Recalling that $\mathcal{T}_i$ is lower triangular, and engaging in some manipulation, we obtain

$$\sum_i \xi_i^T \xi_i = \sum_i (t_{x,i} - l_{x,i})^2 + (t_{y,i} - l_{y,i})^2$$

where

$$\begin{aligned} l_{x,i} &= p_{x,i} + p_{x,i}(k_1(p_{x,i}^2 + p_{y,i}^2) + k_2(p_{x,i}^2 + p_{y,i}^2)^2) + p_1(p_{x,i}^2 + p_{y,i}^2 + 2p_{x,i}^2) + 2p_2 p_{x,i} p_{y,i} \\ l_{y,i} &= p_{y,i} + p_{y,i}(k_1(p_{x,i}^2 + p_{y,i}^2) + k_2(p_{x,i}^2 + p_{y,i}^2)^2) + p_2(p_{x,i}^2 + p_{y,i}^2 + 2p_{y,i}^2) + 2p_1 p_{x,i} p_{y,i} \end{aligned}$$

$$\begin{aligned} p_{x,i} &= \frac{i_{11} g_{x,i} + i_{12} g_{y,i} + i_{13} g_{i,3}}{g_{i,3}} \\ p_{y,i} &= \frac{i_{22} g_{x,i} + i_{23} g_{i,3}}{g_{i,3}} \end{aligned}$$

and

$$\begin{aligned} g_{x,i} &= e_{11} s_{x,i} + e_{12} s_{y,i} + e_{13} s_{z,i} + e_{14} \\ g_{y,i} &= e_{21} s_{x,i} + e_{22} s_{y,i} + e_{23} s_{z,i} + e_{24} \\ g_{z,i} &= e_{31} s_{x,i} + e_{32} s_{y,i} + e_{33} s_{z,i} + e_{34} \end{aligned}$$

(which you should check as an exercise). This is a constrained optimization problem, because $\mathcal{T}_e$ is a Euclidean transformation. The constraints here are

$$\begin{aligned} 1 - \sum_v e_{j,1v}^2 = 0 \text{ and } 1 - \sum_v e_{j,2v}^2 = 0 \text{ and } 1 - \sum_v e_{j,3v}^2 = 0 \\ \sum_v e_{j,1v} e_{j,2v} = 0 \text{ and } \sum_v e_{j,1v} e_{j,3v} = 0 \text{ and } 1 - \sum_v e_{j,2v} e_{j,3v} = 0 \end{aligned} .$$

We might just throw this into a constrained optimizer (review Section 32.2), but good behavior requires a good start point. This can be obtained by a little manipulation, which I work through in the next section. Some readers may prefer to skip this at first (or even higher) reading because it's somewhat specialized, but it shows how the practical application of some tricks worth knowing.