CSE 252B: Imaging Geometry

Lecturer: Ben Ochoa

Scribed by Owen Jow on January 07, 2019

1 Imaging Geometry

The focus of this course is **imaging geometry**, particularly involving multiple views. Essentially, we want to understand the geometry that links 2D images to the 3D world, and 2D images to other 2D images of the same 3D scene. With this knowledge, we can perform e.g. 3D reconstruction from images (which is probably the main application space where this course is concerned).

We will start with the case of one view (one image), but ultimately will study scenarios in which we have anywhere from 1 to n views of a scene. We'll also study both the *calibrated* and *uncalibrated* settings. In the calibrated case, we know the internal camera properties (intrinsics) such as focal length and principal point (e.g. via EXIF tags) and are able to reconstruct 3D up to a scaling factor. In the [more challenging] uncalibrated case, we know nothing about the cameras and can only perform 3D reconstruction up to a projective transformation[1] unless we estimate the intrinsics. If we work in projective space everything will be very distorted, but the math checks out.

Note that in spite of the name "imaging geometry," we really want to drop the "image" part as quickly as possible. Instead of images, we want to work with geometric primitives such as points, hyperplanes, and hyperquadrics.[2] We also want to work with the duals of these things.[3]

To get to geometric primitives, we typically extract distinct, photo-identifiable *feature points* from each image. Again, note that each 2D feature point will correspond to both a 3D point (2D-3D correspondence) *and* 2D points in other views (2D-2D correspondence). In this class, we will use simple feature detectors/matchers in order to learn how to deal with noise (i.e. how to perform outlier rejection based on imaging geometry constraints).

We use the points and correspondences to estimate some model, e.g. a geometric transformation.

- **Euclidean (rigid body) transformation**: a rotation and translation. (3 DOF)[4]
- **Similarity transformation**: a rotation, scaling, and translation. (4 DOF)[5]
- **Affine transformation**: a linear transformation[6] and translation. (6 DOF)[7]
- **Projective transformation (homography)**: 3×3 homogeneous matrix. (8 DOF)[8]

[1] But then, using auto-calibration, we can transform our reconstruction in projective space into metric space.

[2] *Hyperquadrics*: a generalization of conics in n-dimensional space. (In 3D, conics are quadrics.)

[3] Just as points and hyperplanes are duals of each other, hyperquadrics have duals too.

[4] [assumes 2D case] angle for 2D rotation (1 DOF), translation in two dimensions (2 DOF)

[5] [assumes 2D case] Euclidean (3 DOF) plus scale factor (1 DOF)

[7] [assumes 2D case] 2×2 linear transformation matrix (4 DOF), translation in two dimensions (2 DOF)

[8] [assumes 2D case] 3×3 matrix (9 DOF) minus one DOF because "up to scale"

1.1 Projection

Formally, projection involves the transformation of an entity from one dimensionality to another dimensionality, where the second dimensionality is less than or equal to the first (e.g. 3D $\rightarrow$ 3D, or 3D $\rightarrow$ 2D). *Back-projection* involves a transformation from a lower dimensionality to a higher dimensionality. However, there is an ambiguity in doing this (e.g. a 2D image point's analogue in the world can lie anywhere along the 3D ray passing through the point and the center of projection).

To resolve this ambiguity, we normally need more than one image. Then, e.g. in stereo geometry, we can *triangulate* – compute the 3D point where the rays intersect. In practice, the rays will be noisy and will miss each other, so we'll have to do some kind of optimization (looking for the point in 3D which minimizes distance between the rays, or which minimizes reprojection error in all images).

1.2 Nonlinear Optimization: Initialization

Typically, the success of nonlinear optimization depends a lot on the initialization. We'll generally want the parameters to be initialized as close as possible to their best selves, so that it's easier to reach that global optimum. To initialize, we'll usually use the estimate given by a linear optimization.

1.3 Miscellaneous

Three-View Geometry	Removes a lot of ambiguity, allows for line-based reconstruction.
Stratified Reconstruction	Go from projective to affine to similarity reconstruction.
Auto-Calibration	Determine intrinsics from image geometry alone.

2 Homogeneous Coordinates: Notation

In effect, "homogeneous" means "defined up to a scale." Before being introduced to homogeneous coordinates, one generally uses *inhomogeneous* representations, which will be marked notationally with tildes, e.g. in 2D an inhomogeneous point is $\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y})^T$. By contrast, homogeneous coordinates will be unmarked, e.g. in 2D a homogeneous point will be written $\mathbf{x} = (x, y, w)^T$ where w is the "projective coordinate." Then $\tilde{\mathbf{x}} = (x/w, y/w)^T$ and $\mathbf{x} = (\lambda\tilde{x}, \lambda\tilde{y}, \lambda)^T$.

One of the draws of homogeneous coordinates is that they allow us to have meaningful points at infinity which are transformed back and forth between non-infinite and infinite representations.

Positive infinity is defined

$$\infty = \lim_{x \to 0^+} \frac{1}{x}$$

whereas negative infinity is

$$-\infty = \lim_{x \to 0^-} \frac{1}{x}$$

Similarly, zero can be defined

$$0 = \lim_{x \to \infty} \frac{1}{x}$$

In homogeneous coordinates, a point at infinity has a zero projective coordinate. For example, the set of 2D points at infinity is $\mathbf{x}_\infty = (x, y, 0)^T$. The set of 3D points at infinity is $\mathbf{X}_\infty = (X, Y, Z, 0)^T$.