EE 5885 - Geometry and Deep Computer Vision Notes

Libao Jin

January 17, 2020

Contents

1 Representation of a Three-Dimensional Moving Scene

1.1 Three-dimensional Euclidean Space

Euclidean space, which is a set of elements satisfying the five axioms of Euclid, is denoted by $\mathbb{E}^3$. Analytically, three-dimensional Euclidean space can be represented globally by a Cartesian coordinate frame: every point $p \in \mathbb{E}^3$ can be identified with a point in $\mathbb{R}^3$ with three coordinates

$$\mathbf{X} \doteq \begin{bmatrix} X_1 & X_2 & X_3 \end{bmatrix}^T = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \in \mathbb{R}^3.$$

Definition 1.1 (Vector)**.** In Euclidean space, a vector is determined by a pair of points $p, q \in \mathbb{E}^3$ and is defined as a directed arrow connecting p to q, denoted $v = \overrightarrow{pq}$, p is called the base point of the vector v. If p has coordinates $\mathbf{X}$ and q has coordinates $\mathbf{Y}$, then $v = \overrightarrow{pq} \doteq \mathbf{Y} - \mathbf{X} \in \mathbb{R}^3$. More specifically, this kind of vector is referred to as a *bound vector*. On the other hand, a vector which does not depend on its base point is called a *free vector*.

Definition 1.2 (Linear vector space)**.** The set of all free vectors forms a *linear vector space*, with the linear combination of two vectors $u, v \in \mathbb{R}^3$ defined by

$$\alpha u + \beta v = \begin{bmatrix} \alpha u_1 + \beta v_1 & \alpha u_2 + \beta v_2 & \alpha u_3 + \beta v_3 \end{bmatrix}^T \in \mathbb{R}^3, \forall \alpha, \beta \in \mathbb{R}.$$

Definition 1.3 (Inner product)**.** The *standard Euclidean metric* for $\mathbb{E}^3$ is defined by an inner product on the vector space $\mathbb{R}^3$, which can be converted to the following canonical form

$$\langle u, v \rangle \doteq u^T v = u_1 v_1 + u_2 v_2 + u_3 v_3, \forall u, v \in \mathbb{R}^3.$$

When the inner product between two vectors is zero, i.e. $\langle u, v \rangle = 0$, they are said to be *orthogonal*.

Definition 1.4 (Cross product)**.** Given two vectors $u, v \in \mathbb{R}^3$, their *cross product* is a third vector with coordinates given by

$$u \times v \doteq \widehat{u} v = \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} u_2 v_3 - u_3 v_2 \\ u_3 v_1 - u_1 v_3 \\ u_1 v_2 - u_2 v_1 \end{bmatrix} \in \mathbb{R}^3,$$

where $\widehat{u}$ is a skew-symmetric matrix, i.e., $\widehat{u}^T = -\widehat{u}$.

Proposition 1.1 (Properties of inner product and cross product)**.**

(a) $\langle u \times v, u \rangle = \langle u \times v, v \rangle = 0$.
(b) $u \times v = -v \times u$.

Definition 1.5 (Tangent vector)**.** If the trajectory of a moving particle p in $\mathbb{E}^3$ is described by a curve $\gamma(\cdot) : t \mapsto \mathbf{X}(t) \in \mathbb{R}^3, t \in [0, 1]$, then the total length of the curve is given by

$$l(\gamma(\cdot)) = \int_0^1 \|\dot{\mathbf{X}}(t)\| \, dt,$$

where $\dot{\mathbf{X}}(t) = d\mathbf{X}(t)/dt \in \mathbb{R}^3$ is the so-called *tangent vector* to the curve.

Lemma 1.1 (Skew-symmetric matrix)**.** A matrix $M \in \mathbb{R}^{3\times 3}$ is skew-symmetric if and only if $M = \widehat{u}$ for some $u \in \mathbb{R}^3$. In other words, the vector space $\mathbb{R}^3$ and the space of all skew-symmetric 3×3 matrices ($so(3)$) are isomorphic (i.e., there exists a one-to-one map that preserves the vector space structure). The isomorphism is the so-called *hat operator* $\widehat{\ }: \mathbb{R}^3 \to so(3); u \mapsto \widehat{u}$, and its inverse map, called the *vee operator*, which extracts the components of the vector u from a skew-symmetric matrix $\widehat{u}$, is given by $\vee : so(3) \to \mathbb{R}^3; \widehat{u} \mapsto u$.

1.2 Rigid-body motion

Definition 1.6 (Rigid-body motion)**.** A *rigid-body motion* (or rigid-body transformation) is then a family of maps that describe how the coordinates of every point on a rigid object change in time while satisfying

$$\|\mathbf{X}(t) - \mathbf{Y}(t)\| \equiv \text{ constant}, \forall t \in \mathbb{R},$$

where $\mathbf{X}(t)$ and $\mathbf{Y}(t)$ are the coordinates of any two points p and q on the object, respectively. We denote such a map by

$$g(t) : \mathbb{R}^3 \to \mathbb{R}^3; \mathbf{X} \mapsto g(t)(\mathbf{X}).$$

If instead of looking at the entire continuous path of the moving object, we concentrate on the map between its initial and final configuration, we have a *rigid-body displacement*, denoted by

$$g : \mathbb{R}^3 \to \mathbb{R}^3; \mathbf{X} \mapsto g(\mathbf{X}).$$

Suppose that v is a vector defined by points p and q with coordinates $v = \mathbf{Y} - \mathbf{X}$; then, after the transformation g, we obtain a new vector

$$u = g_*(v) \doteq g(\mathbf{Y}) - g(\mathbf{X}).$$

Since g preserves the distance between points, we have that $\|g_*(v)\| = \|v\|$ for all free vectors $v \in \mathbb{R}^3$.

Definition 1.7 (Euclidean transformation)**.**

(a) A map that preserves the distance is called a *Euclidean transformation*.
(b) The map or transformation induced by a rigid-body motion is called a *special Euclidean transformation* (orientation-preserving).

Definition 1.8 (Rigid-body motion or special Euclidean transformation)**.** A map $g : \mathbb{R}^3 \to \mathbb{R}^3$ is a rigid-body motion or a special Euclidean transformation if it preserves the norm and the cross product of any two vectors.

(a) norm: $\|g_*(v)\| = \|v\|, \forall v \in \mathbb{R}^3$, which implies a rigid-body motion preserves inner product, because $\langle u, v\rangle = \frac{1}{4}(\|u+v\|^2 - \|u-v\|^2)$. One can conclude that, for any rigid-body motion g,

$$\langle u, v\rangle = \langle g_*(u), g_*(v)\rangle, \forall u, v \in \mathbb{R}^3.$$

(b) cross product: $g_*(u) \times g_*(v) = g_*(v \times v), \forall u, v \in \mathbb{R}^3$.

The collection of all such motions or transformations is denoted by $SE(3)$.

As a consequence, a rigid-body motion can be described by the motion of a chosen point on the body and the rotation of a coordinate frame attached to that point. We represent the *configuration* of a rigid body by attaching a Cartesian coordinate frame to some point on the rigid body, and we will keep track of the motion of this coordinate frame relative to a fix *world (reference) frame*. Consider a coordinate frame, with its principal axes given by three *orthonormal* vectors $e_1, e_2, e_3 \in \mathbb{R}^3$; that is, they satisfy

$$e_i^T e_j = \delta_{ij} \doteq \begin{cases} 1 & \text{for } i = j, \\ 0 & \text{for } i \neq j. \end{cases}$$

The vectors are ordered so as to form a right-handed frame: $e_1 \times e_2 = e_3$. Then, after a rigid-body motion g, we have

$$g_*(e_i)^T g_*(e_j) = \delta_{ij}, g_*(e_1) \times g_*(e_2) = g_*(e_3).$$

That is, the resulting three vectors $g_*(e_1)$, $g_*(e_2)$, $g_*(e_3)$ still form a right-handed orthonormal frame. Therefore, a rigid object can always be associated with a right-handed orthonormal frame, which we call the *object coordinate frame* or the *body coordinate frame*, and its rigid-body motion can be entirely specified by the motion of such a frame.
Given a world reference frame $W : (X, Y, Z)$, pick a fixed point o on the camera and attach it to an object frame, the configuration of the camera relative to the world frame W can be determined by two components:

(a) the vector between the camera frame $C : (x, y, z)$ and that of the camera frame, $g(o)$, called the "translational" part and denoted by T;
(b) the relative orientation of the camera frame C, with coordinate axes (x, y, z), relative to the fixed world frame W with coordinate axes (X, Y, Z), called the "rotational" part and denoted by R.

1.3 Rotational motion and its representation

Without loss of generality, assume that the origin of the world frame is the center of rotation o.
The configuration (or "orientation") of the frame C relative to the frame W is determined by the coordinate of the three orthonormal vectors

$$r_1 = g_*(e_1), r_2 = g_*(e_2), r_3 = g_*(e_3) \in \mathbb{R}^3$$

relative to the world frame W. The three vectors r_1, r_2, r_3 are simply the unit vectors along the three principal axes x, y, z of the frame C, respectively. The configuration of the rotation object is then completely determined by the 3×3 matrix

$$R_{wc} \doteq \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix} \in \mathbb{R}^3,$$

with r_1, r_2, r_3 stacked in order as its three columns. Since r_1, r_2, r_3 form an orthonormal frame, it follows that

$$r_i^T r_j = \delta_{ij} \doteq \begin{cases} 1 & \text{for } i = j, \\ 0 & \text{for } i \neq j, \end{cases} \forall i, j \in \{1, 2, 3\}.$$

This can be written matrix form as

$$R_{wc}^T R_{wc} = R_{wc} R_{wc}^T = I.$$

Any matrix that satisfies the above identity is called the *orthogonal matrix*, whose inverse is its transpose. Besides, $R_{wc} \in SO(3)$ represents the coordinate transformation from the frame C to the frame W. Let $\mathbf{X}_w = \begin{bmatrix} X_{1w} & X_{2w} & X_{3w} \end{bmatrix}^T \in \mathbb{R}^3$, $\mathbf{X}_c = \begin{bmatrix} X_{1c} & X_{2c} & X_{3c} \end{bmatrix}^T \in \mathbb{R}^3$, then

$$\mathbf{X}_w = R_{wc}\mathbf{X}_c \iff \mathbf{X}_c = R_{wc}^{-1}\mathbf{X}_w = R_{wc}^T\mathbf{X}_w = R_{cw}\mathbf{X}_w.$$

Definition 1.9 (Special orthogonal matrix)**.** The space of all special (orientation-preserving) orthogonal matrices in $\mathbb{R}^{3\times3}$ is denoted by

$$SO(3) \doteq \{R \in \mathbb{R}^{3\times3} | R^T R = I, \det(R) = 1\}.$$

Traditionally, 3×3 special orthogonal matrices are called *rotation matrices*. $SO(3)$ is also referred to as the *special orthogonal group* of $\mathbb{R}^3$, or simply the *rotation group*.

The configuration of a continuously rotating object can be described as a trajectory $R(t) : t \mapsto SO(3)$ in the space $SO(3)$. The relative motion between time t_2 and time t_1 will be denoted as $R(t_2, t_1)$. The composition law of the rotation group implies

$$R(t_2, t_0) = R(t_2, t_1)R(t_1, t_0), \forall t_0 < t_1 < t_2 \in \mathbb{R}.$$

For a rotating camera, the world coordinates $\mathbf{X}_w$ of a fixed 3-D point p are transformed to its coordinates relative to the camera frame C by

$$\mathbf{X}_c(t) = R_{cw}(t)\mathbf{X}_w.$$

Alternatively, if a point p is fixed with respect to the camera frame has coordinates $\mathbf{X}_c$, its world coordinates $\mathbf{X}_w(t)$ as a function of t are then given by

$$\mathbf{X}_w(t) = R_{wc}(t)\mathbf{X}_c.$$

1.3.1 Canonical exponential coordinates for rotations

Given a trajectory $R(t) : \mathbb{R} \to SO(3)$ that describes a continuous rotational motion, the rotation must satisfy the following constraint

$$R(t)R^T(t) = I \implies \dot{R}(t)R^T(t) + R(t)\dot{R}^T(t) = 0 \implies \dot{R}(t)R^T(t) = -(\dot{R}(t)R^T(t))^T.$$

That is, $\dot{R}(t)R^T(t) \in \mathbb{R}^{3\times 3}$ is a skew-symmetric matrix, by Lemma 1.1, there must exist a vector, say $\omega(t) \in \mathbb{R}^3$, such that

$$\dot{R}(t)R^T(t) = \widehat{\omega}(t) \implies \dot{R}(t) = \widehat{\omega}(t)R(t).$$

Assume that $\widehat{\omega}$ is constant, then $R(t)$ can be interpreted as the *state transition matrix* for the linear ordinary differential equation:

$$\dot{x}(t) = \widehat{\omega}x(t), x(t) \in \mathbb{R}^3 \implies x(t) = e^{\widehat{\omega}t}x(0),$$

where $e^{\widehat{\omega}t}$ is the matrix exponential

$$e^{\widehat{\omega}t} = I + \widehat{\omega}t + \frac{(\widehat{\omega}t)^2}{2!} + \cdots + \frac{(\widehat{\omega}t)^n}{n!} + \cdots$$

Assuming that $R(0) = I$ is the initial condition for the differential equation, we must have

$$R(t) = e^{\widehat{\omega}t}.$$

A *physical interpretation of the above equation* is that if $\|\omega\| = 1$, then $R(t) = e^{\widehat{\omega}t}$ is simply a rotation around the axis $\omega \in \mathbb{R}^3$ by an angle of t radians.

Definition 1.10 (Tangent space)**.** The space of all skew-symmetric matrices is denoted by

$$so(3) \doteq \{\widehat{\omega} \in \mathbb{R}^{3\times 3} | \omega \in \mathbb{R}^3\}.$$

The matrix exponential indeed defines a map from the space $so(3)$ to $SO(3)$, the so-called *exponential map*

$$\exp : so(3) \to SO(3); \widehat{\omega} \mapsto e^{\widehat{\omega}}.$$

Theorem 1.1 (Logarithm of $SO(3)$)**.** For any $R \in SO(3)$, there exists a (not necessarily unique) $\omega \in \mathbb{R}^3$ such that $R = \exp(\widehat{\omega})$. We denote the inverse of the exponential map by $\widehat{\omega} = \log(R)$.

Proof. The proof is by construction: if the notation matrix $R \neq I$ is given as

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix},$$

the corresponding ω is given by

$$\|\omega\| = \cos^{-1}\left(\frac{\mathrm{tr}(R)-1}{2}\right), \frac{\omega}{\|\omega\|} = \frac{1}{2\sin(\|\omega\|)}\begin{bmatrix} r_{32}-r_{23} \\ r_{13}-r_{31} \\ r_{21}-r_{12} \end{bmatrix}.$$

If $R = I$, then $\|\omega\| = 0$, and $\omega/\|\omega\|$ is not determined (and therefore can be chosen arbitrarily). □

Theorem 1.2 (Rodrigue's formula for a rotation matrix)**.** Given $\omega \in \mathbb{R}^3$, the matrix exponential $R = e^{\widehat{\omega}}$ is given by

$$e^{\widehat{\omega}} = I + \frac{\widehat{\omega}}{\|\omega\|}\sin(\|\omega\|) + \frac{\widehat{\omega}^2}{\|\omega\|^2}(1-\cos(\|\omega\|)).$$

Remark 1.1. In general, the difference between $\widehat{\omega}_1\widehat{\omega}_2$ and $\widehat{\omega}_2\widehat{\omega}_1$ is called the Lie bracket on $so(3)$, denoted by

$$[\widehat{\omega}_1, \widehat{\omega}_2] = \widehat{\omega}_1\widehat{\omega}_2 - \widehat{\omega}_2\widehat{\omega}_1, \forall \widehat{\omega}_1, \widehat{\omega}_2 \in so(3).$$

From the definition above it can be verified that $[\widehat{\omega}_1, \widehat{\omega}_2]$ is also a skew-symmetric matrix in $so(3)$. The linear structure of $so(3)$ together with the Lie bracket form the Lie algebra of the (Lie) group $SO(3)$. Given $\widehat{\omega}$, the set of all rotation matrices $e^{\widehat{\omega}t}, t \in \mathbb{R}$, is then a one-parameter subgroup of $SO(3)$, i.e., the planar rotation group $SO(2)$. The multiplication in such a subgroup is always commutative, since for the same $\omega \in \mathbb{R}^3$, we have

$$e^{\widehat{\omega}t_1}e^{\widehat{\omega}t_2} = e^{\widehat{\omega}t_2}e^{\widehat{\omega}t_1} = e^{\widehat{\omega}(t_1+t_2)}, \forall t_1, t_2 \in \mathbb{R}.$$

1.4 Rigid-body motion and its representation

To describe the coordinates of a point p on the object with respect to the world frame W, assume that $\mathbf{X}_c$ are coordinates of the point p relative to the frame C, with respect to the world frame W, it becomes $R_{wc}\mathbf{X}_c$, where $R_{wc} \in SO(3)$ is the relative rotation between two frames. Hence, the coordinates $\mathbf{X}_w$ are given by

$$\mathbf{X}_w = R_{wc}\mathbf{X}_c + T_{wc}.$$

Denote the full rigid-body motion by $g_{wc} = (R_{wc}, T_{wc})$, or simply $g = (R, T)$ if the frames involved are clear from the context. In compact form, we write

$$\mathbf{X}_w = g_{wc}(\mathbf{X}_c).$$

Definition 1.11 (Special Euclidean transformations)**.** The set of all possible configurations of a rigid body can be described by the space of rigid-body motions or special Euclidean transformations

$$SE(3) \doteq \{g = (R, T) | R \in SO(3), T \in \mathbb{R}^3\} = \left\{\bar{g} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} | R \in SO(3), T \in \mathbb{R}^3\right\}.$$

1.4.1 Homogeneous representation

In contrast to pure rotation case, the coordinate transformation for a full rigid-body motion is not linear but *affine*. We can convert an affine transformation to a linear one by using *homogeneous coordinates* by appending a "1" to the coordinates $\mathbf{X} = \begin{bmatrix} X_1 & X_2 & X_3 \end{bmatrix} \in \mathbb{R}^3$ of a point $p \in \mathbb{E}^3$ yields a vector in $\mathbb{R}^4$, denoted by

$$\bar{\mathbf{X}} \doteq \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix} = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ 1 \end{bmatrix} \in \mathbb{R}^4.$$

In effect, such an extension of coordinates has embedded the Euclidean space $\mathbb{E}^3$ into a hyperplane in $\mathbb{R}^4$ instead of $\mathbb{R}^3$. Homogeneous coordinates of a vector $v = \mathbf{X}(q) - \mathbf{X}(p)$ are defined as the difference between homogeneous coordinates of the two points hence of the form

$$\bar{v} \doteq \begin{bmatrix} v \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{X}(q) \\ 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}(p) \\ 1 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{bmatrix} \in \mathbb{R}^4.$$

Using the new notation, the (affine) transformation can then be rewritten in a "linear" form

$$\bar{\mathbf{X}}_w = \begin{bmatrix} \mathbf{X}_w \\ 1 \end{bmatrix} = \begin{bmatrix} R_{wc} & T_{wc} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}_c \\ 1 \end{bmatrix} \doteq \bar{g}_{wc} \bar{\mathbf{X}}_c,$$

where the 4×4 matrix $\bar{g}_{wc} \in \mathbb{R}^{4\times4}$ is called the *homogeneous representation* of the rigid-body motion $g_{wc} = (R_{wc}, T_{wc}) \in SE(3)$. In general, if $g = (R, T)$, then its homogeneous representation is

$$\bar{g} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times4}.$$

In particular, $\forall g_1, g_2$ and $g \in SE(3)$, we have

$$\bar{g}_1 \bar{g}_2 = \begin{bmatrix} R_1 & T_1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_2 & T_2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_1 R_2 & R_1 T_2 + T_1 \\ 0 & 1 \end{bmatrix} \in SE(3)$$

and

$$\bar{g}^{-1} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} R^T & -R^T T \\ 0 & 1 \end{bmatrix} \in SE(3).$$

in the homogeneous representation, the action of a rigid-body motion $g \in SE(3)$ on a vector $v = \mathbf{X}(q) - \mathbf{X}(p) \in \mathbb{R}^3$ becomes

$$\bar{g}_*(\bar{v}) = \bar{g}\bar{\mathbf{X}}(q) - \bar{g}\bar{\mathbf{X}}(p) = \bar{g}\bar{v}.$$

1.4.2 Canonical exponential coordinates for rigid-body motions

Consider the motion of a continuously moving rigid body described by a trajectory on $SE(3)$: $g(t) = (R(t), T(t))$, or in the homogeneous representation

$$g(t) = \begin{bmatrix} R(t) & T(t) \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times4}.$$

In analogy with the case of a pure rotation, consider the structure of the matrix

$$\dot{g}(t)g^{-1}(t) = \begin{bmatrix} \dot{R}(t)R^T(t) & \dot{T}(t) - \dot{R}(t)R^T(t)T(t) \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4}.$$

It is known that $\dot{R}(t)R^T(t)$ is a skew-symmetric matrix; i.e., there exists $\widehat{\omega}(t) \in so(3)$ such that $\widehat{\omega}(t) = \dot{R}(t)R^T(t)$. Define a vector $v(t) \in \mathbb{R}^3$ such that $v(t) = \dot{T}(t) - \widehat{\omega}(t)T(t)$. Then the above equation becomes

$$\dot{g}(t)g^{-1}(t) = \begin{bmatrix} \widehat{\omega}(t) & v(t) \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4}.$$

Define a matrix $\widehat{\xi} \in \mathbb{R}^{4\times 4}$ as follows

$$\widehat{\xi}(t) = \dot{g}(t)g^{-1}(t) = \begin{bmatrix} \widehat{\omega}(t) & v(t) \\ 0 & 0 \end{bmatrix},$$

then we have

$$\dot{g}(t) = [\dot{g}(t)g^{-1}(t)]g(t) = \widehat{\xi}(t)g(t),$$

where $\widehat{\xi}$ can be viewed as the "tangent vector" along the curve of $g(t)$ and can be used to approximate $g(t)$ locally:

$$g(t + dt) \approx g(t) + \widehat{\xi}(t)g(t)dt = [I + \widehat{\xi}(t)dt]g(t).$$

A 4×4 matrix of the form of $\widehat{\xi}$ is called a *twist*. The set of all twists is denoted by

$$se(3) \doteq \left\{ \widehat{\xi} = \begin{bmatrix} \widehat{\omega} & v \\ 0 & 0 \end{bmatrix} \middle| \widehat{\omega} \in so(3), v \in \mathbb{R}^3 \right\} \subset \mathbb{R}^{4\times 4}.$$

The set $se(3)$ is called the tangent space (or Lie algebra) of the matrix group $SE(3)$. We also define two operators "$\vee$" and "$\wedge$" to convert between a twist $\widehat{\xi} \in se(3)$ and its *twist coordinates* $\xi \in \mathbb{R}^6$ as follows:

$$\begin{bmatrix} \widehat{\omega} & v \\ 0 & 0 \end{bmatrix}^{\vee} \doteq \begin{bmatrix} v \\ \omega \end{bmatrix} \in \mathbb{R}^6, \begin{bmatrix} v \\ \omega \end{bmatrix}^{\wedge} \doteq \begin{bmatrix} \widehat{\omega} & v \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4}.$$

In the twist coordinate ξ, we refer to v as the *linear velocity* and ω as the *angular velocity*, which indicates that they are related to either the translational or the rotational part of the full motion. Suppose that the twist $\widehat{\xi}$ is a constant matrix, we have the special case

$$\dot{g}(t) = \widehat{\xi}g(t) \implies g(t) = e^{\widehat{\xi}t}g(0).$$

Assuming the initial condition $g(0) = I$, we may conclude that

$$g(t) = e^{\widehat{\xi}t},$$

where the twist exponential is

$$e^{\widehat{\xi}t} = I + \widehat{\xi}t + \frac{(\widehat{\xi}t)^2}{2!} + \cdots + \frac{(\widehat{\xi}t)^n}{n!} + \cdots.$$

By Rodrigue's formula, we can obtain

$$e^{\widehat{\xi}} = \begin{bmatrix} e^{\widehat{\omega}} & \frac{(I - e^{\widehat{\omega}})\widehat{\omega}v + \omega\omega^T v}{\|\omega\|} \\ 0 & 1 \end{bmatrix}, \text{if } \omega \neq 0.$$

The exponential map defines a transformation from the space $se(3)$ to $SE(3)$,

$$\exp : se(3) \to SE(3); \widehat{\xi} \mapsto e^{\widehat{\xi}},$$

and the twist $\widehat{\xi} \in se(3)$ is also called the *exponential coordinates* for $SE(3)$, as $\widehat{\omega} \in so(3)$ for $SO(3)$.

Theorem 1.3 (Logarithm of $SE(3)$)**.** For any $g \in SE(3)$, there exist (not necessarily unique) twist coordinates $\xi = (v, w)$ such that $g = \exp(\widehat{\xi})$. We denote the inverse to the exponential map by $\widehat{\xi} = \log(g)$.

Proof. The proof is constructive. Suppose $g = (R, T)$. From Theorem 1.1, for the rotation matrix $R \in SO(3)$ we can always find ω such that $e^{\widehat{\omega}} = R$. If $R \neq I$, i.e., $\|\omega\| \neq 0$, then we can solve for $v \in \mathbb{R}^3$ from the linear equation

$$\frac{(I - e^{\widehat{\omega}})\widehat{\omega}v + \omega\omega^T v}{\|\omega\|} = T.$$

If $R = I$, then $\|\omega\| = 0$. In this case, we may simply choose $\omega = 0, v = T$. □

Remark 1.2. As in the rotation case, the linear structure of $se(3)$, together with the closure under the Lie bracket operation

$$[\widehat{\xi}_1, \widehat{\xi}_2] = \widehat{\xi}_1\widehat{\xi}_2 - \widehat{\xi}_2\widehat{\xi}_1 = \begin{bmatrix} \widehat{\omega_1 \times \omega_2} & \omega_1 \times v_2 - \omega_2 \times v_1 \\ 0 & 0 \end{bmatrix} \in se(3),$$

makes $se(3)$ the Lie algebra for $SE(3)$. The two rigid-body motions $g_1 = e^{\widehat{\xi}_1}$ and $g_2 = e^{\widehat{\xi}_2}$ commute with each other, $g_1 g_2 = g_2 g_1$, if and only if $[\widehat{\xi}_1, \widehat{\xi}_2] = 0$.

1.4.3 Coordinate and velocity transformations

(a) Rules of coordinate transformations. The time $t \in \mathbb{R}$ will typically be used to index camera motion, we will use $g(t) = (R(t), T(t)) \in SE(3)$ or

$$g(t) = \begin{bmatrix} R(t) & T(t) \\ 0 & 0 \end{bmatrix} \in SE(3)$$

to denote the relative displacement between some fixed world frame W and the camera frame C at time $t \in \mathbb{R}$. By default, we assume $g(0) = I$, i.e., at time $t = 0$ the camera frame coincides with the world frame. So if the coordinates of a point $p \in \mathbb{E}^3$ relative to the world frame are $\mathbf{X}_0 = \mathbf{X}(0)$, its coordinates relative to the camera at time t are given by

$$\mathbf{X}(t) = R(t)\mathbf{X}_0 + T(t),$$

or in the homogeneous representation,

$$\mathbf{X}(t) = g(t)\mathbf{X}_0.$$

If the camera is at locations $g(t_1), g(t_2), \ldots, g(t_m)$ at times $t_1, t_2, \ldots, t_m$, respectively, then the coordinates of the point p are given as $\mathbf{X}(t_i) = g(t_i)\mathbf{X}_0, i = 1, 2, \ldots, m$, correspondingly. If it is only the position, not the time, that matters, we will often use g_i as a shorthand for $g(t_i)$, and similarly R_i for $R(t_i)$, T_i for $T(t_i)$, and $\mathbf{X}_i$ for $\mathbf{X}(t_i)$. We hence have

$$\mathbf{X}_i = R_i\mathbf{X}_0 + T_i.$$

Composition of rigid-body motions:

$$\mathbf{X}(t_3) = g(t_3, t_2)\mathbf{X}_2 = g(t_3, t_2)g(t_2, t_1)X(t_1) = g(t_3, t_1)\mathbf{X}(t_1) \implies g(t_3, t_1) = g(t_3, t_2)g(t_2, t_1),$$

and

$$g(t_2, t_1)g(t_1, t_2) = g(t_2, t_2) = I \implies g^{-1}(t_2, t_1) = g(t_1, t_2).$$

Use g_{ij} as a shorthand for $g(t_i, t_j)$, the composition rule then become

$$\mathbf{X}_i = g_{ij}\mathbf{X}_j, g_{ik} = g_{ij}g_{jk}, g_{ij}^{-1} = g_{ji}.$$

(b) Rules of velocity transformation. It is known that the coordinates $\mathbf{X}(t)$ of a point $p \in \mathbb{E}^3$ relative to a moving camera are a function of time t:

$$\mathbf{X}(t) = g_{cw}(t)\mathbf{X}_0.$$

Then velocity of the point p relative to the (instantaneous) camera frame is

$$\dot{\mathbf{X}}(t) = \dot{g}_{cw}(t)\mathbf{X}_0.$$

In order to express $\dot{\mathbf{X}}(t)$ in terms of quantities in the moving frame, we substitute $\mathbf{X}_0$ by $g_{cw}^{-1}(t)\mathbf{X}(t)$ and, using the notion of twist, define

$$\widehat{V}_{cw}^c(t) = \dot{g}_{cw}(t)g_{cw}^{-1}(t) \in se(3),$$

then

$$\dot{\mathbf{X}}(t) = \widehat{V}_{cw}^c(t)\mathbf{X}(t).$$

Suppose that a viewer is in another coordinate frame displaced relative to the camera frame by rigid-body transformation $g \in SE(3)$. Then the coordinates of the same point p relative to this frame are $\mathbf{Y}(t) = g\mathbf{X}(t)$. We compute the velocity in the new frame, and obtain

$$\dot{\mathbf{Y}}(t) = g\dot{g}_{cw}(t)g_{cw}^{-1}(t)g^{-1}\mathbf{Y}(t) = g\widehat{V}_{cw}^c g^{-1}\mathbf{Y}(t).$$

So the new velocity (or twist) is

$$\widehat{V} = g\widehat{V}_{cw}^c g^{-1},$$

which is the same physical quantity but viewed from a different vantage pint. Use the *adjoint map* on the space $se(3)$:

$$ad_g : se(3) \to se(3); \widehat{\xi} \mapsto g\widehat{\xi}g^{-1},$$

we have $\widehat{V} = ad_g(\widehat{V}_{cw}^c)$.

2 Image Formation

In a broad figurative sense, vision is the inverse problem of image formation: the latter studies how objects give rise to images, while the former attempts to use images to recover a description of objects in space.

2.1 Representation of images

Definition 2.1 (Image)**.** An *image* is a two-dimensional brightness array. In other words, it is a map I, defined on a compact region Ω of a two-dimensional surface, taking values in the positive real numbers. For instance, in the case of a camera, Ω is a planar, rectangular region occupied by the photographic medium or by the CCD sensor. So I is a function

$$I : \Omega \subset \mathbb{R}^2 \to \mathbb{R}_+; (x, y) \mapsto I(x, y).$$

Such a value $I(x, y)$ is typically called *image intensity* or *brightness*, or more formally *irradiance*, which has the units of power per unit area ($\mathrm{W/m^2}$)

Definition 2.2 (Picture)**.** A *picture* can be thought of as a scene different from the true one that produces on the imaging sensor (the eye in this case) the same image as the true one.

2.2 Lenses, light, and basic photometry

2.2.1 Imaging through lenses

A camera (or in general an optical system) is composed of a set of lenses used to "direct" light. By directing light we mean a controlled change in the direction of propagation, which can be performed by means of diffraction, refraction, and reflection. We will consider only the simplest possible model, that of a *thin lens*, which is a mathematical model defined by an axis (*optical axis*), and a plane perpendicular to the axis (*focal plane*), with a circular aperture centered at the *optical center*, i.e., the intersection of the focal plane with the optical axis. The thin lens has two parameters: its *focal length* f and its *diameter* d. Its function is characterized by two properties:

(a) All rays entering the aperture parallel to the optical axis intersect on the optical axis at a distance f from the optical center. The point of intersection is called the *focus* of the lens.
(b) All rays through the optical center are undeflected.

Proposition 2.1 (Fundamental Equation of the Thin Lens)**.** Let p be a point and the point x the image of the point p, under the assumption of a thin lens, the irradiance $I(\mathbf{x})$ at the point $\mathbf{x}$ with coordinates (x, y) on the image plane is obtained by integrating all the energy emitted from the region of space contained in the cone determined by the geometry of the lens.

$$\frac{1}{Z} + \frac{1}{z} = \frac{1}{f}.$$

2.2.2 Imaging through a pinhole

If we let the aperture of a thin lens decrease to zero, all rays are forced to go through the optical center o, and therefore they remain undeflected. Consequently, the aperture of the cone decreases to zero, and the only points that contribute to the irradiance at the image point $\mathbf{x} = \begin{bmatrix} x & y \end{bmatrix}^T$ are on a line through the center o of the lens. If a point p has coordinates $\mathbf{X} = \begin{bmatrix} X & Y & Z \end{bmatrix}^T$ relative to a reference frame centered at the optical center o, with its z-axis being the optical axis (of the lens), then we have the so-called ideal *perspective projection*:

$$x = -f\frac{X}{Z}, y = -f\frac{Y}{Z},$$

where f is referred to as the *focal length*. We simply write the projection as a map π:

$$\pi : \mathbb{R}^3 \to \mathbb{R}^2; \mathbf{X} \mapsto \mathbf{x}.$$

This imaging model is called *ideal pinhole camera model*.

Definition 2.3 (Field of view)**.** *Field of view* (FOV) is the angle subtended by the spatial extent of the sensor as seen from the optical center. If $2r$ is the largest spatial extension of the sensor (e.g., the side of the CCD), then the field of view is $\theta = 2\arctan(r/f)$.

2.3 A geometric model of image formation

Under the assumptions of a pinhole camera model and Lambertian surfaces, one can essentially reduce the process of image formation to tracing rays from points on objects to pixels. That is, knowing which point in space projects onto which point on the image plane allows one to directly associate the radiance at the point to the irradiance of its images. A mathematical model accounts for three types of transformations is used:

(a) coordinate transformation between the camera frame and the world frame;
(b) projection of 3-D coordinate onto 2-D image coordinates;
(c) coordinate transformation between possible choices of image coordinate frame.

We will describe such a (simplified) image formation process as a series of transformations of coordinates. Inverting such a chain of transformations is generally referred to as "camera calibration".

2.3.1 An ideal perspective camera

Consider a generic point p, with coordinates $\mathbf{X}_0 = \begin{bmatrix} X_0 & Y_0 & Z_0 \end{bmatrix}^T \in \mathbb{R}^3$ relative to the world reference frame, while the coordinates $\mathbf{X} = \begin{bmatrix} X & Y & Z \end{bmatrix}^T$ of the same point p relative to the camera frame are given by a rigid-body transformation $g = (R, T)$ of $\mathbf{X}_0$:

$$\mathbf{X} = R\mathbf{X}_0 + T \in \mathbb{R}^3.$$

Adopting the frontal pinhole camera model, the point $\mathbf{X}$ is projected onto the image plane at the point

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} = \frac{f}{Z} \begin{bmatrix} X \\ Y \end{bmatrix}.$$

In homoogeneous coordinates, this relationship can be written as

$$Z \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \iff Z\mathbf{x} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{X}.$$

Since the coordinate Z (or the depth of the point p) is usually unknown, we may simply write it as an arbitrary positive scalar $\lambda \in \mathbb{R}_+$. Notice that, we can decompose the matrix into

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Define two matrices,

$$K_f \doteq \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{3\times 3}, \Pi_0 \doteq \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \in \mathbb{R}^{3\times 4}.$$

The matrix Π_0 is referred to as the *standard (or "canonical") projection matrix*. To summarize, the overall geometric model for *an ideal camera* can be described as

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{bmatrix} \iff \lambda\mathbf{x} = K_f \Pi_0 g \mathbf{X}_0.$$

If the focal length f is known and hence can be normalized to 1, this model reduces to a Euclidean transformation g followed by a standard projection Π_0, i.e.,

$$\lambda\mathbf{x} = \Pi_0 g \mathbf{X}_0.$$

2.3.2 Camera with intrinsic parameters

The ideal model is specified relative to the "canonical retinal frame", centered at the optical center with one axis aligned with the optical axis. In practice, when one captures images with a digital camera the measurements are obtained in terms of pixels (i, j), with the origin of the image coordinate frame typically in the upper-left corner of the image. We need to specify the relationship between the retinal plane coordinate frame and the pixel array. We need to specify the units along the x- and y-axes: if (x, y) are specified in terms of metric units (e.g., millimeters), and (x_s, y_s) are scaled versions that correspond to coordinates of the pixel, then the transformation can be described by a scaling matrix

$$\begin{bmatrix} x_s \\ y_s \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

that depends on the size of the pixel (in metric units) along the x and y directions. When $s_x = s_y$, each pixel is square. Here x_s and y_s are specified relative to the *principal point* (where the z-axis intersects the image plane), whereas the pixel index (i, j) is conventionally specified relative to the upper-left corner, and is indicated by positive numbers. Therefore, we need to translate the origin of the reference frame to this corner,

$$\begin{aligned} x' &= x_s + o_x, \\ y' &= y_s + o_y, \end{aligned}$$

where (o_x, o_y) are the coordinates (in pixels) of the principal point relative to the image reference frame. In the homogeneous representation, we have

$$\mathbf{x}' \doteq \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix},$$

where x' and y' are actual image coordinates in pixels. In case the pixels are not rectangular, a more general form of the scaling matrix can be considered,

$$\begin{bmatrix} s_x & s_\theta \\ 0 & s_y \end{bmatrix} \in \mathbb{R}^{2\times 2},$$

where s_θ is called a *skew factor* and is proportional to $\cot\theta$, where θ is the angle between the image axes x_s and y_s. The transformation matrix takes the general form

$$K_s \doteq \begin{bmatrix} s_x & s_\theta & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{3\times 3}.$$

A more realistic model of a transformation between homogeneous coordinates of a 3-D point relative to the camera frame and homogeneous coordinates of its image expressed in terms of pixels is as follows,

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & s_\theta & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{bmatrix} \iff \lambda \mathbf{x}' = K_s K_f \Pi_0 g \mathbf{X}_0.$$

Let

$$K \doteq K_s K_f \doteq \begin{bmatrix} s_x & s_\theta & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} fs_x & fs_\theta & o_x \\ 0 & fs_y & o_y \\ 0 & 0 & 1 \end{bmatrix}.$$

Then we have

$$\lambda \mathbf{x}' = K \Pi_0 g \mathbf{X}_0.$$

The constant 3×4 matrix Π_0 represents the perspective projection. The upper triangular 3×3 matrix K collects all parameters that are "intrinsic" to a particular camera, and is therefore called the *intrinsic parameter matrix*, or the *calibration matrix* of the camera. The entries of the matrix K have the following geometric interpretations:

- o_x: x-coordinate of the principal point in pixels,
- o_y: y-coordinate of the principal point in pixels,
- $fs_x = \alpha_x$: size of unit length in horizontal pixels,
- $fs_y = \alpha_y$: size of unit length in vertical pixels,
- α_x/α_y: aspect ratio σ,
- fs_θ: skew of the pixel, often close to zero.

To summarize, the geometric relationship between a point of coordinates $\mathbf{X}_0 = \begin{bmatrix} X_0 & Y_0 & Z_0 & 1 \end{bmatrix}^T$ relative to the world frame and its corresponding image coordinates $\mathbf{x}' = \begin{bmatrix} x' & y' & 1 \end{bmatrix}^T$ (in pixels) depends on the rigid-body motion (R, T) between the world frame and the camera frame (sometimes referred to as the *extrinsic calibration parameters*), an ideal projection Π_0, and the camera intrinsic parameter K. The overall model for image formation is therefore captured by the following equation:

$$\lambda \mathbf{x}' = K \Pi_0 g \mathbf{X}_0 = \begin{bmatrix} KR & KT \end{bmatrix} \mathbf{X}_0 = \Pi \mathbf{X}_0. \tag{2.1}$$

At this stage, in order to explicitly see the nonlinear nature of the perspective projection equation, we can divide (2.1) by the scale λ and obtain the following expressions for the image coordinates (x', y', z'),

$$x' = \frac{\pi_1^T \mathbf{X}_0}{\pi_3^T \mathbf{X}_0}, y' = \frac{\pi_2^T \mathbf{X}_0}{\pi_3^T \mathbf{X}_0}, z' = 1,$$

where $\pi_1^T, \pi_2^T, \pi_3^T \in \mathbb{R}^4$ are the three rows of the projection matrix Π. For convenience, we often write $\mathbf{x} \sim \mathbf{y}$ for two (homogeneous) vectors $\mathbf{x}$ and $\mathbf{y}$ equal up to a scalar factor.

2.3.3 Radial distortion

In addition to linear distortions described by the parameters in K, if a camera with a wide field of view is used, one can often observe significant distortion along radial direction. The simplest effective model for such a distortion is:

$$x = x_d(1 + a_1 r^2 + a_2 r^4),$$
$$y = y_d(1 + a_1 r^2 + a_2 r^4),$$

where (x_d, y_d) are coordinates of the distorted points, $r^2 = x_d^2 + y_d^2$ and a_1, a_2 are additional camera parameters that model the amount of distortion. In case the calibration rig is not available, the radial distortion parameters can be estimated directly from images. A simple method is that assumes a more general model of radial distortion:

$$\mathbf{x} = c + f(r)(\mathbf{x}_d - c),$$
$$f(r) = 1 + a_1 r + a_2 r^2 + a_3 r^3 + a_4 r^4,$$

where $\mathbf{x}_d = \begin{bmatrix} x_d & y_d \end{bmatrix}^T$ are the distorted image coordinates, $r^2 = \|\mathbf{x}_d - c\|^2, c = \begin{bmatrix} c_x & c_y \end{bmatrix}^T$ is the center of the distortion, not necessarily coincident with the center of the image, and $f(r)$ is the distortion correction factor.

2.3.4 Image, preimage, and coimage of points and lines

To avoid possible confusion that can be caused by different representations for the same geometric entity, we introduce a few abstract notions related to the image of a point or a line. Consider the perspective projection of a straight line L onto the 2-D image plane. To specify a line in 3-D, we can typically specify a point p_o, called the base point, on the line and specify a vector v that indicates the direction of the line. Suppose that $\mathbf{X}_o = \begin{bmatrix} X_o & Y_o & Z_o & 1 \end{bmatrix}^T$ are the homogeneous coordinates of the base point p_o and $\mathbf{V} = \begin{bmatrix} V_1 & V_2 & V_3 & 0 \end{bmatrix}^T \in \mathbb{R}^4$ is the homogeneous representation of v, relative to the camera coordinate frame. Then (homogeneous) coordinates of any point on the line L can be expressed as

$$\mathbf{X} = \mathbf{X}_o + \mu \mathbf{V}, \mu \in \mathbb{R}.$$

Then, the image of the line L is given by the collection of image points with homogeneous coordinates given by

$$\mathbf{x} \sim \Pi_0 \mathbf{X} = \Pi_0(\mathbf{X}_o + \mu \mathbf{V}) = \Pi_0 \mathbf{X}_o + \mu \Pi_0 \mathbf{V}.$$

Definition 2.4 (Preimage)**.** A preimage of a point or a line in the image plane is the set of 3-D points that give rise to an image equal to the given point or line.

A plane can be represented by its normal vector.

Definition 2.5 (Coimage)**.** The coimage of a point or a line is defined to be the subspace in $\mathbb{R}^3$ that is the (unique) orthogonal complement of its preimage.

Proposition 2.2. Image, preimage, and coimage are equivalent representations:

(a) image = preimage $\cap$ image plane;
(b) preimage = span(image);
(c) preimage = coimage$^\perp$;
(d) coimage = preimage$^\perp$.

Since the preimae of a line L is a two-dimensional subspace, its coimage is represented as the span of the normal vector to the subspace. The notation we use for this is $\ell = \begin{bmatrix} a & b & c \end{bmatrix}^T \in \mathbb{R}^3$. If $\mathbf{x}$ is the image of a point p on this line, then it satisfies the orthogonality equation

$$\ell^T \mathbf{x} = 0.$$

Table 1: The image, preimage, and coimage of a point and a line

Notation	Image	Preimage	Coimage
Point	span($\mathbf{x}$) $\cap$ image plane	span($\mathbf{x}$) $\subset \mathbb{R}^3$	span($\widehat{\mathbf{x}}$) $\subset \mathbb{R}^3$
Line	span(ℓ) $\cap$ image plane	span($\widehat{\ell}$) $\subset \mathbb{R}^3$	span(ℓ) $\subset \mathbb{R}^3$

3 Reconstruction from Two Calibrated Views

3.1 Epipolar geometry

Consider two images of the same scene taken from two distinct vantage points. If we assume that the camera is *calibrated*, the homogeneous image coordinates $\mathbf{x}$ and the spatial coordinates $\mathbf{X}$ of a point p, with respect to the camera frame, are related by

$$\lambda \mathbf{x} = \Pi_0 \mathbf{X},$$

where $\Pi_0 = \begin{bmatrix} I & 0 \end{bmatrix}$. That is, the image $\mathbf{x}$ differs from the actual 3-D coordinates of the point by an unknown (depth) scale $\lambda \in \mathbb{R}_+$. For simplicity, we will assume that the scene is *static* and that the position of corresponding feature points across images is available.

3.1.1 The epipolar constraint and the essential matrix

An orthonormal reference frame is associated with each camera, with its origin o at the optical center and the z-axis aligned with the optical axis. The relationship between the 3-D coordinates of a point in the inertial "world" coordinate frame and the camera frame can be expressed by a rigid-body transformation. Without loss of generality, we can assume the world frame to be one of the cameras, while the other is positioned and oriented according to a Euclidean transformation $g = (R, T) \in SE(3)$. If we call the 3-D coordinates of a point p relative to the two camera frames $\mathbf{X}_1 \in \mathbb{R}^3$ and $\mathbf{X}_2 \in \mathbb{R}^3$, they are related by a rigid-body transformation in the following way:

$$\mathbf{X}_2 = R\mathbf{X}_1 + T.$$

Now let $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^3$ be the homogeneous coordinates of the projection of the *same* point p in the two image planes. Since $\mathbf{X}_i = \lambda_i \mathbf{x}_i, i = 1, 2$, this equation can be written in terms of the image coordinates $\mathbf{x}_i$ and the depths λ_i as

$$\lambda_2 \mathbf{x}_2 = R\lambda_1 \mathbf{x}_1 + T.$$

In order to eliminate the depths λ_i in the preceding equation, pre-multiply both sides by $\widehat{T}$ to obtain

$$\lambda_2 \widehat{T}\mathbf{x}_2 = \widehat{T}R\lambda_1 \mathbf{x}_1.$$

Since the vector $\widehat{T}\mathbf{x}_2 = T \times \mathbf{x}_2$ is perpendicular to the vector $\mathbf{x}_2$, the inner product $\langle \mathbf{x}_2, \widehat{T}\mathbf{x}_2 \rangle = \mathbf{x}_2^T \widehat{T}\mathbf{x}_2$ is zero. Then we have the following theorem:

Theorem 3.1 (Epipolar constraint)**.** Consider two images $\mathbf{x}_1$, $\mathbf{x}_2$ of the same point p from two camera positions with relative pose (R, T), where $R \in SO(3)$ is the relative orientation and $T \in \mathbb{R}^3$ is the relative position. Then $\mathbf{x}_1$, $\mathbf{x}_2$ satisfy

$$\langle \mathbf{x}_2, T \times R\mathbf{x}_1 \rangle = 0, \text{ or } \mathbf{x}_2^T \widehat{T}R\mathbf{x}_1 = 0. \tag{3.1}$$

The matrix $E \doteq \widehat{T}R \in \mathbb{R}^{3\times 3}$ is the epipolar constraint equation is called the *essential matrix*, which encodes the relative pose between the two cameras. The epipolar constraint (3.1) is also called the *essential constraint*. Since the epipolar constraint is bilinear in each of its arguments $\mathbf{x}_1$ and $\mathbf{x}_2$, it is also called the *bilinear constraint*.

Definition 3.1 (Epipolar geometric entities)**.**

(a) The plane (o_1, o_2, p) determined by the two centers of projection o_1, o_2 and the point p is called an *epipolar plane* associated with the camera configuration and point p. There is one epipolar plane for each point p.
(b) The projection $\mathbf{e}_1(\mathbf{e}_2)$ of one camera center onto the image plane of the other camera frame is called an *epipole*. Note that the projection may occur outside the physical boundary of the imaging sensor.
(c) The intersection of the epipolar plane of p with one image plane is a line $\ell_1(\ell_2)$, which is called the epipolar line of p. We usually use the normal vector $\ell_1(\ell_2)$ to the epipolar plane to denote this line.

Proposition 3.1 (Properties of episodes and epipolar lines)**.** Given an essential matrix $E = \widehat{T}R$ that defines an epipolar relation between two images $\mathbf{x}_1$, $\mathbf{x}_2$, we have:

(a) The two epipoles $\mathbf{e}_1, \mathbf{e}_2 \in \mathbb{R}^3$, with respect to the first and second camera frames, respectively, are the left and right null spaces of E, respectively:

$$\mathbf{e}_2^T E = 0, E\mathbf{e}_1 = 0.$$

That is, $\mathbf{e}_2 \sim T$ and $\mathbf{e}_1 \sim R^T T$. We recall that $\sim$ indicates equality up to a scalar factor.
(b) The (coimages of) epipolar lines $\ell_1, \ell_2 \in \mathbb{R}^3$ associated with the two image points $\mathbf{x}_1$, $\mathbf{x}_2$ can be expressed as

$$\ell_2 \sim E\mathbf{x}_1, \ell_1 \sim E^T\mathbf{x}_2 \in \mathbb{R}^3,$$

where ℓ_1, ℓ_2 are in fact the normal vectors to the epipolar plane expressed with respect to the two camera frames, respectively.
(c) In each image, both the image point and the epipole lie on the epipolar line

$$\ell_i^T \mathbf{e}_i = 0, \ell_i^T \mathbf{x}_i = 0, i = 1, 2.$$

3.1.2 Elementary properties of the essential matrix

The matrix $E = \widehat{T}R \in \mathbb{R}^{3\times3}$ contains information about the relative position T and orientation $R \in SO(3)$ between the two cameras. Matrices of this form belong to a very special set of matrices in $\mathbb{R}^{3\times3}$ called the *essential space* and denoted by $\mathcal{E}$:

$$\mathcal{E} \doteq \{\widehat{T}R | R \in SO(3), T \in \mathbb{R}^3\} \subset \mathbb{R}^{3\times3}.$$

Lemma 3.1 (The hat operator)**.** For a vector $T \in \mathbb{R}^3$ and a matrix $K \in \mathbb{R}^{3\times3}$, if $\det(K) = 1$ and $T' = KT$, then $\widehat{T} = K^T\widehat{T}'K$.

Theorem 3.2 (Characterization of the essential matrix)**.** A nonzero matrix $E \in \mathbb{R}^{3\times3}$ is an essential matrix if and only if E has a singular value decomposition (SVD) $E = U\Sigma V^T$ with

$$\Sigma = \text{diag}\{\sigma, \sigma, 0\}$$

for some $\sigma \in \mathbb{R}_+$ and $U, V \in SO(3)$.

Proof. Choose $U = R_0^T R_Z(\pi/2)$, $V^T = R_0 R$, then

$$E = \widehat{T}R = R_0^T R_Z(\pi/2)\,\text{diag}\{\|T\|, \|T\|, 0\}R_0 R.$$

□

Lemma 3.2. Consider an arbitrary nonzero skew-symmetric matrix $\widehat{T} \in so(3)$ with $T \in \mathbb{R}^3$. If for a rotation matrix $R \in SO(3)$, $\widehat{T}R$ is also a skew-symmetric matrix, then $R = I$ or $R = e^{\widehat{u}\pi}$, where $u = T/\|T\|$. Further, $\widehat{T}e^{\widehat{u}\pi} = -\widehat{T}$.

Theorem 3.3 (Pose recovery from the essential matrix)**.** There exist exactly two relative poses (R, T) with $R \in SO(3)$ and $T \in \mathbb{R}^3$ corresponding to a nonzero essential matrix $E \in \mathcal{E}$.

3.2 Basic reconstruction algorithm

3.2.1 The eight-point linear algorithm

Let $E = \widehat{T}R$ be the essential matrix associated with the epipolar constraint. The entries of this 3×3 matrix are denoted by

$$E = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

and stacked into a vector $E^s \in \mathbb{R}^9$, which is typically referred to as the stacked version of the matrix E:

$$E^s = \begin{bmatrix} e_{11} & e_{21} & e_{31} & e_{12} & e_{22} & e_{32} & e_{13} & e_{23} & e_{33} \end{bmatrix}^T \in \mathbb{R}^9,$$

The inverse operation from E^s to its matrix version is then called *unstacking*. We further denote the *Kronecker product* $\otimes$ of two vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ by

$$\mathbf{a} \doteq \mathbf{x}_1 \otimes \mathbf{x}_2.$$

Or, more specifically, if $\mathbf{x}_1 = \begin{bmatrix} x_1 & y_1 & z_1 \end{bmatrix}^T \in \mathbb{R}^3$ and $\mathbf{x}_2 = \begin{bmatrix} x_2 & y_2 & z_2 \end{bmatrix}^T \in \mathbb{R}^3$, then

$$\mathbf{a} = \begin{bmatrix} x_1x_2 & x_1y_2 & x_1z_2 & y_1x_2 & y_1y_2 & y_1z_2 & z_1x_2 & z_1y_2 & z_1z_2 \end{bmatrix}^T \in \mathbb{R}^9.$$

Since the epipolar constraint $\mathbf{x}_2^T E \mathbf{x}_2 = 0$ is linear in the entries of E, using the above notation we can rewrite it as the inner product of $\mathbf{a}$ and E^s:

$$\mathbf{a}^T E^s = 0.$$

Given a set of corresponding image points $\begin{bmatrix} \mathbf{x}_1^j & \mathbf{x}_2^j \end{bmatrix}^T, j = 1, 2, \ldots, n$, define a matrix $\chi \in \mathbb{R}^{n\times 9}$ associated with these measurements to be

$$\chi \doteq \begin{bmatrix} \mathbf{a}^1 & \mathbf{a}^2 & \cdots & \mathbf{a}^n \end{bmatrix}^T,$$

where the jth row $\mathbf{a}^j$ is the Kronecker product of each pair $\begin{bmatrix} \mathbf{x}_1^j & \mathbf{x}_2^j \end{bmatrix}$. The absence of noise, the vector E^s satisfies

$$\chi E^s = 0.$$

Theorem 3.4 (Projection onto the essential space)**.** Given a real matrix $F \in \mathbb{R}^{3\times 3}$ with SVD $F = U \operatorname{diag}\{\lambda_1, \lambda_2, \lambda_3\} V^T$ with $U, V \in SO(3), \lambda_1 \geq \lambda_2 \geq \lambda_3$, then the essential matrix $E \in \mathcal{E}$ that minimizes the error $\|E - F\|_f^2$ is given by $E = U \operatorname{diag}\{\sigma, \sigma, 0\} V^T$ with $\sigma = (\lambda_1 + \lambda_2)/2$. The subscript "$f$" indicated the Frobenius norm of a matrix. This is the square norm of the sum of the squares of all the entries of the matrix.

Algorithm 1: The eight-point algorithm

Function eight-point($\mathbf{x}_1^j, \mathbf{x}_2^j$)**:**

Input: A set of image correspondences $(\mathbf{x}_1^j, \mathbf{x}_2^j), j = 1, 2, \ldots, n (n \geq 8)$.

Output: $(R, T) \in SE(3)$ such that ${\mathbf{x}_2^j}^T \widehat{T} R \mathbf{x}_1^j = 0, j = 1, 2, \ldots, n$.

(a) Compute a first approximation of the essential matrix. Construct $\chi = \begin{bmatrix} \mathbf{a}^1 & \mathbf{a}^2 & \cdots & \mathbf{a}^n \end{bmatrix}^T \in \mathbb{R}^{n \times 9}$ from correspondences $\mathbf{x}_1^j$ and $\mathbf{x}_2^j$, namely,

$$\mathbf{a}^j = \mathbf{x}_1^j \otimes \mathbf{x}_2^j \in \mathbb{R}^9.$$

Find the vector $E^s \in \mathbb{R}^9$ of unit length such that $\|\chi E^s\|$ is minimized as follows: compute the SVD of $\chi = U_\chi \Sigma_\chi V_\chi^T$ and define E^s to be the ninth column of V_χ. Unstack the nine elements of E^s into a square 3×3 matrix E. Note that this matrix will in general *not* be in the essential matrix.

(b) Project onto the essential space. Compute the singular value decomposition of the matrix E recovered from data to be

$$E = U \operatorname{diag}\{\sigma_1, \sigma_2, \sigma_3\} V^T,$$

where $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq 0$ and $U, V \in SO(3)$. In general, since E may not be an essential matrix, $\sigma_1 \neq \sigma_2$ and $\sigma_3 \neq 0$. But its projection onto the normalized essential space is $U \Sigma V^T$, where $\Sigma = \operatorname{diag}\{1, 1, 0\}$.

(c) Recover the displacement from the essential matrix. We now need only U and V to extract R and T from the essential matrix as

$$R = U R_Z^T(\pm\pi/2) V^T, \widehat{T} = U R_Z(\pm\pi/2) \Sigma U^T,$$

where $R_Z^T(\pm\pi/2) = \begin{bmatrix} 0 & \pm 1 & 0 \\ \mp 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$.

end

3.3 The Optimal Quaternion

A quaternion q is a 4-dim, unit vector (i.e., $q^T q = 1$) that can express rotation,

$$\mathbf{q} = \begin{bmatrix} q_0 \\ q \end{bmatrix}.$$

If $R = e^{\widehat{\omega}\theta}$, then $q = \omega \sin(\theta/2)$, $q_0 = \cos(\theta/2)$. Also:

Theorem 3.5.

$$v^T R z = q^T D q + v^T z,$$

where

$$D = \begin{bmatrix} 0 & (\widehat{z}v)^T \\ \widehat{z}v & zv^T + vz^T - 2(v^T z) I \end{bmatrix}_{4 \times 4}.$$

Proof.

$$R = I + 2q_0\widehat{q} + 2\widehat{q}^2 = I + 2q_0\widehat{q} + 2(qq^T - q^T q I).$$

Then we have

$$\begin{aligned} v^T Rz &= v^T[I + 2q_0\widehat{q} + 2(qq^T - q^T qI)]z \\ &= v^T z + 2q_0 v^T \widehat{q}z + 2v^T(qq^T z) - 2v^T q^T qIz \\ &= v^T z - 2q_0 v^T \widehat{z}q + 2q^T vz^T q - 2q^T(v^T zI)q. \end{aligned}$$

Consider the quadratic

$$\begin{bmatrix} \mathbf{x}_1^T & \mathbf{x}_2^T \end{bmatrix} \begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \mathbf{x}_1^T A\mathbf{x}_1 + 2\mathbf{x}_1^T B\mathbf{x}_2 + \mathbf{x}_2^T C\mathbf{x}_2.$$

□

This can be used to estimate a rotation matrix from a set of corresponding 3-D vectors. Let R be the unknown rotation matrix that rotates vectors z^i (known) so they are aligned with vectors v^i (known also). You want to find the best R such that $v^i = Rz^i$ for $i = 1, 2, \ldots, n$. Let

$$\begin{aligned} J &= \sum_{i=1}^{n} \|Rz^i - v^i\|_2^2 \\ &= \sum_{i=1}^{n} (Rz^i - v^i)^T (Rz^i - v^i) \\ &= \sum_{i=1}^{n} (z^{iT} R^T - v^{iT})(Rz^i - v^i) \\ &= \sum_{i=1}^{n} z^{iT} R^T Rz^i - z^{iT} R^T v^i - v^{iT} Rz^i + v^{iT} v^i \\ &= \sum_{i=1}^{n} z^{iT} z^i - v^{iT} Rz^i - v^{iT} Rz^i + v^{iT} v^i \\ &= \sum_{i=1}^{n} z^{iT} z^i - 2v^{iT} Rz^i + v^{iT} v^i \\ &= \sum_{i=1}^{n} (-2q^T D^i q) + \sum_{i=1}^{n} (z^{iT} z^i + v^{iT} v^i) - 2v^{iT} z^i \\ &= -2q^T \left(\sum_{i=1}^{n} D^i \right) q + \sum_{i=1}^{n} (z^{iT} z^i + v^{iT} v^i) - 2v^{iT} z^i \\ &= -2q^T Dq + \sum_{i=1}^{n} (z^{iT} z^i + v^{iT} v^i) - 2v^{iT} z^i. \end{aligned}$$

where

$$D^i = \begin{bmatrix} 0 & (\widehat{z}^i v^i)^T \\ \widehat{z}^i v^i & z^i v^{iT} + v^i z^{iT} - 2z^{iT} v^i I \end{bmatrix}.$$

We want to minimize J with respect to q, but $q^T q = 1$. Use Lagrange multipliers. Let

$$L = J + \nu(1 - q^T q) = -2q^T Dq + \sum_{i=1}^{n} (z^{iT} z^i + v^{iT} v^i) - 2v^{iT} z^i + \nu(1 - q^T q).$$

Now we can set derivative of L with respect to q and ν to zero:

$$\frac{\partial L}{\partial q} = -4Dq + 2\nu q = 0 \implies Dq = \frac{\nu}{2}q.$$

Therefore, q is an eigenvector of D and $\nu/2$ is its eigenvalue. It is known that D is a 4×4 matrix, so it has four eigenvectors. Which minimizes J? Therefore,

Algorithm 2:

Function ():

Input:

Output:

(a) Measure v^i and z^i. Want to find R so that $v^i = Rz^i$.

(b) $D^i = \begin{bmatrix} 0 & (\widehat{z}^i v^i)^T \\ \widehat{z}^i v^i & z^i v^{iT} + v^i z^{iT} - 2z^{iT} v^i I \end{bmatrix}$.

(c) $D = \sum_{i=0}^{n} D^i$.

(d) Find the eigenvectors and eigenvalues of D.

(e) $\mathbf{q}$ is the eigenvector which corresponds to the maximal eigenvalue.

(f) Convert $\mathbf{q}$ to R, $R = I + 2q_0\widehat{q} + 2(qq^T - q^T qI)$.

return

end

3.3.1 Estimating R When a Calibrated Camera is purely Rotating

Let $\mathbf{x}_1'$ and $\mathbf{x}_2'$ be the pixel coordinates, then we have the following relation

$$\lambda_2 \mathbf{x}_2' = R\lambda_1 \mathbf{x}_1' + T.$$

Since it's purely rotating, i.e., $T = 0$. Therefore, $\lambda_2 \mathbf{x}_2' = R\lambda_1 \mathbf{x}_1'$. Rotating a vector does not change its length. Let

$$\mathbf{u}_2 = \frac{\lambda_2 \mathbf{x}_2'}{\|\lambda_2 \mathbf{x}_2\|} = \frac{\mathbf{x}_2'}{\|\mathbf{x}_2'\|}, \mathbf{u}_1 = \frac{\mathbf{x}_1'}{\|\mathbf{x}_1'\|} \implies \mathbf{u}_2 = R\mathbf{u}_1.$$

Take n points, then

$$\mathbf{u}_2^i = R\mathbf{u}_1^i, i = 1, 2, \ldots, n.$$

Now the optimal quaternion algorithm can be used to estimate R.

3.3.2 Estimating K and R: Uncalibrated Camera is Purely Rotating

Given the pixel coordinate $\mathbf{x}'$ and the calibration matrix K, we can calculate the coordinate on the image plane $\mathbf{x}$ as follows,

$$\mathbf{x}' = K\mathbf{x} \implies \mathbf{x} = K^{-1}\mathbf{x}' \implies \mathbf{x}_1 = K_1^{-1}\mathbf{x}_1', \mathbf{x}_2 = K_2^{-1}\mathbf{x}_2'.$$

It is known that $\lambda_2 \mathbf{x}_2 = R\lambda_1 \mathbf{x}_1$, we have

$$\lambda_2 K_2^{-1}\mathbf{x}_2' = \lambda_1 R K_1^{-1}\mathbf{x}_1' \implies \lambda_2 \mathbf{x}_2' = \lambda_1 K_2 R K_1^{-1}\mathbf{x}_1'.$$

Assume that $K_1 = K_2 = K$, then $\lambda_2^i(\mathbf{x}_2^i)' = \lambda_1^i KRK^{-1}(\mathbf{x}_1^i)'$ for $i = 1, 2, \ldots, n$. If K was known, R could be found from the optimal quaternion algorithm, search numerically for K to minimize reprojection error.

3.3.3 Reprojection Error In Image 2

Suppose estimates of K and R have been found, named K_e and R_e.

$$\lambda_2 \mathbf{x}_2' = KRK^{-1}\lambda_1 \mathbf{x}_1' \iff \lambda_1 \mathbf{x}_1' = KR^TK^{-1}\lambda_1 \mathbf{x}_2'.$$

If you know K, R, and $\mathbf{x}_1'$, you can find $\mathbf{x}_2'$. We can reproject $\mathbf{x}_1$ into image 2 using our estimated K and R,

$$\lambda_2 \mathbf{x}_{2'}' = \lambda_1 K_e R_e K_e^{-1}\mathbf{x}_1' \text{ or } \mathbf{x}_{2'}' \sim K_e R_e K_e^{-1}\mathbf{x}_1',$$

The total reprojection error in image 2 is then

$$J_2 = \sum_{i=1}^{n} \|(\mathbf{x}_2^i)' - (\mathbf{x}_{2'}^i)'\| = \sum_{i=1}^{n} \|(\mathbf{x}_2^i)' - \lambda_1 K_e R_e K_e^{-1}(\mathbf{x}_1^i)'\|.$$

We can also reproject $\mathbf{x}_2'$ into image 1 using the same estimated K and R,

$$\lambda_1 \mathbf{x}_{1'}' = \lambda_2 K_e R_e^T K_e^{-1}\mathbf{x}_2' \text{ or } \mathbf{x}_{1'}' \sim K_e R_e^T K_e^{-1}\mathbf{x}_2', i = 1, 2, \ldots, n.$$

The total reprojection error in image 1 is then

$$J_1 = \sum_{i=1}^{n} \|(\mathbf{x}_1^i)' - (\mathbf{x}_{1'}^i)'\| = \sum_{i=1}^{n} \|(\mathbf{x}_1^i)' - \lambda_2 K_e R_e^T K_e^{-1}(\mathbf{x}_2^i)'\|.$$

3.3.4 Find K and R

If K was known, R could be found from the optimal quaternion algorithm, search numerically for K to minimize reprojection error.

Algorithm 3:

Function ():

Input: $(\mathbf{x}_1^i)', (\mathbf{x}_2^i)', i = 1, 2, \ldots, n.$

Output:

(a) Pick K.

(b) Use Optimal Quaternion Algorithm to find R.

(c) Calculate reprojections,

$$\mathbf{x}_{2'}' = KRK^{-1}\mathbf{x}_1', \mathbf{x}_{1'}' = KR^TK^{-1}\mathbf{x}_2'.$$

(d) Calculate total reprojetion error:

$$J = J_1 + J_2 = \sum_{i=1}^{n} \|(x_1^i)' - (\mathbf{x}_{1'}^i)'\|^2 + \sum_{i=1}^{n} \|(\mathbf{x}_2^i)' - (\mathbf{x}_{2'}^i)'\|^2.$$

(e) Repeat the steps above until a K is found that gives a small error (J).

$$K = \begin{bmatrix} k_1 & k_2 & k_3 \\ 0 & k_4 & k_5 \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{k} = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \\ k_5 \end{bmatrix}.$$

end

3.3.5 Calibrating From A Sequence of Rotations

If a sequence of image all have the same calibration matrix, K, then they *all* be used to estimate K. Let R_j be the rotation matrix from $j-1$th system to jth system $R_{j,j-1}$.

$$\lambda_2 \mathbf{x}_2 = R\lambda_1 \mathbf{x}_1, \lambda_2 \mathbf{x}_2' = K_2 R \lambda_1 (K_1^{-1} \mathbf{x}_1').$$

If $K_1 = K_2 = \cdots = K_m$, then $\lambda_j K^{-1} \mathbf{x}_j' = R_j \lambda_{j-1} K^{-1} \mathbf{x}_j'$. Again, if we...

$$J_j = \sum_{i=1}^{n} \{\|(\mathbf{x}_j^i)' - (\mathbf{x}_{j'}^i)'\| - \|(\mathbf{x}_{j-1}^i)' - (\mathbf{x}_{j-1'}^i)'\|\}$$

In MATLAB, `cpselect` can be used to manually put in correspondence points. (In our project, we used SIFT to get corresponding points automatically)

3.3.6 Finding Changing Focal Length in a Sequence of Images From a Purely Rotating Camera

Suppose a camera is purely rotating ($T = 0$), and the focal length changes between taking images (auto focus). Find $K_1, K_2, K_3, \ldots$ if K_s is the same and known for all images.

$$K = K_s K_f.$$

Assume that K_s is constant and already found.

$$\mathbf{x} = K\mathbf{X} = K_s K_f \mathbf{X} \implies K_s^{-1}\mathbf{x} = K_f \mathbf{X} = \mathbf{x}^h.$$

Multiply known pixel locations by known K_s^{-1} to get $\mathbf{x}^h$. Then

$$\mathbf{X} = K_f^{-1}\mathbf{x}^h = \begin{bmatrix} 1/f & 0 & 0 \\ 0 & 1/f & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x}^h.$$

Then we have

$$\mathbf{X}_1 = K_{f1}^{-1}\mathbf{x}_1^h, \mathbf{X}_2 = K_{f2}^{-1}\mathbf{x}_2^h, \mathbf{X}_2 \sim R\mathbf{X}_1 \implies K_{f2}^{-1}\mathbf{X}_2 \sim R K_{f1}^{-1}\mathbf{X}_1.$$

If f_1 and f_2 are known, then R can be found from optimal quaternion algorithm.

(a) Guess f_1 and f_2 (both positive).

$$\mathbf{X}_j^i = K_{f_j}^{-1}\mathbf{x}_j^{ih}, u_j^i = \frac{\mathbf{x}_j^i}{\|\mathbf{x}_j^i\|}, u_j^i = R_j u_{j-1}^i.$$

(b) Use Optimal Quaternion Algorithm to estimate R.
(c) Calculate $J_{\text{repro}} = J_1$.
(d) Go to the first step until J_1 is small.

If a third image is available, $K_{f_3}^{-1}\mathbf{x}_3^h \sim R_3 K_{f_2}^{-1}\mathbf{x}_2^h$. So

(a) Guess f_1, f_2, f_3.
(b) Use optimal quaternion algorithm to estimate $R_2 = R_{21}$ and $R_3 = R_{32}$.
(c) Calculate $J = J_1 + J_2$ which is the reprojection error.
(d) Repeat until J is small.

3.4 Planar scenes and homography

The eight-point algorithm requires the feature points in $3-D$ being in general position, otherwise, say in 2-D plane, it will fail to find a unique solution.

3.4.1 Planar homography

Consider two images of points p on a 2-D plane P in 3-D space. Suppose that two images $(\mathbf{x}_1, \mathbf{x}_2)$ are given for a point $p \in P$ with respect to two camera frames. Let the coordinate transformation between the two frames be

$$\mathbf{X}_2 = R\mathbf{X}_1 + T,$$

where $\mathbf{X}_1, \mathbf{X}_2$ are the coordinates of p relative to camera frames 1 and 2, respectively. The two images $\mathbf{x}_1$, $\mathbf{x}_2$ of p satisfy the epipolar constraint

$$\mathbf{x}_2^T E\mathbf{x}_1 = \mathbf{x}_2^T \widehat{T} R\mathbf{x}_1 = 0.$$

Let $N = \begin{bmatrix} n_1 & n_2 & n_3 \end{bmatrix}^T \in \mathbb{S}^2$ be the unit normal vector of the plane P with respect to the first camera frame, and let $d > 0$ denote the distance from the plane P to the optical center of the first camera. Then we have

$$N^T\mathbf{X}_1 = n_1X + n_2Y + n_3Z = d \iff \frac{1}{d}N^T\mathbf{X}_1 = 1, \forall \mathbf{X}_1 \in P.$$

Then

$$\mathbf{X}_2 = R\mathbf{X}_1 + T = R\mathbf{X}_1 + T\frac{1}{d}N^T\mathbf{X}_1 = \left(R + \frac{1}{d}TN^T\right)\mathbf{X}_1. = H\mathbf{X}_1,$$

where $H = R + \frac{1}{d}TN^T$ is called the (planar) homography matrix, since it denotes a linear transformation from $\mathbf{X}_1 \in \mathbb{R}^3$ to $\mathbf{X}_2 \in \mathbb{R}^3$. It is known that

$$\lambda_1\mathbf{x}_1 = \mathbf{X}_1, \lambda_2\mathbf{x}_2 = \mathbf{X}_2 \implies \lambda_2\mathbf{x}_2 = \lambda_1 H\lambda x_1 \iff \mathbf{x}_2 \sim H\mathbf{x}_1.$$

The last equation is referred to as a *(planar) homography* mapping induced by a plane P.

Proposition 3.2 (Homography for epipolar lines)**.** Given a homography H (induced by plane P in 3-D) between two images, for any pair of corresponding images $(\mathbf{x}_1, \mathbf{x}_2)$ of a 3-D point p that is not necessarily on P, the associated epipolar lines are

$$\ell_2 \sim \widehat{x_2}H\mathbf{x}_1, \ell_1 \sim H^T\ell_2.$$

3.4.2 Estimating the planar homography matrix

In order to further eliminate the unknown scale, multiplying both sides by the skew-symmetric matrix $\widehat{\mathbf{x}_2} \in \mathbb{R}^{3\times3}$, we obtain the equation,

$$\widehat{\mathbf{x}_2}H\mathbf{x}_1 = 0. \tag{3.2}$$

We call this equation the *planar epipolar constraint*, or also the (planar) homography constraint.

Remark 3.1 (Plane as a critical surface)**.** In the planar case, since $\mathbf{x}_2 \sim H\mathbf{x}_1$, for any vector $u \in \mathbb{R}^3$, we have that $u \times \mathbf{x}_2 = \widehat{u}\mathbf{x}_2$ is orthogonal to $H\mathbf{x}_1$. Hence, we have

$$\mathbf{x}_2^T\widehat{u}H\mathbf{x}_1 = 0, \forall u \in \mathbb{R}^3.$$

That is, $\mathbf{x}_2^T E\mathbf{x}_1 = 0$ for a family of matrices $E = \widehat{u}H \in \mathbb{R}^{3\times3}$ besides the essential matrix $E = \widehat{T}R$. This explains why the eight-point algorithm does not apply to feature points from a planar scene.

Since (3.2) is *linear* in H, by stacking the entries of H as a vector,

$$H^s \doteq \begin{bmatrix} H_{11} & H_{21} & H_{31} & H_{12} & H_{22} & H_{32} & H_{13} & H_{23} & H_{33} \end{bmatrix}^T \in \mathbb{R},$$

we may rewrite (3.2) as

$$\mathbf{a}^T H^s = 0,$$

where the matrix $\mathbf{a} = \mathbf{x}_1 \otimes \widehat{\mathbf{x}_2} \in \mathbb{R}^{9\times 3}$ is the Kronecker product of $\widehat{\mathbf{x}_2}$ and $\mathbf{x}_1$. Since the matrix $\widehat{\mathbf{x}_2}$ is only of rank 2, so is the matrix $\mathbf{a}$. Thus, even though the equation $\widehat{\mathbf{x}_2} H \mathbf{x}_1 = 0$ has three rows, it only imposes two independent constraints on H. With this notation, given n pairs of images $\{(\mathbf{x}_1^j, \mathbf{x}_2^j)\}_{j=1}^n$ from points on the same plane P, by defining $\chi \doteq \begin{bmatrix} \mathbf{a}^1 & \mathbf{a}^2 \cdots \mathbf{a}^n \end{bmatrix}^T \in \mathbb{R}^{3n\times 9}$, we may rewrite (3.2) as

$$\chi H^s = 0.$$

Proposition 3.3 (Four-point homography)**.** We have $\text{rank}(\chi) = 8$ if and only if there exists a set of four points (out of the n) such that no three of them are collinear; i.e., they are in a general configuration in the plane.

Thus, if there are more than four image correspondences of which no three in each image are collinear, we may apply standard linear least-squares estimation to find $\min \|\chi H^s\|^2$ to recover H up to a scalar factor. That is, we are able to recover H of the form

$$H_L \doteq \lambda H = \lambda \left(R + \frac{1}{d} T N^T \right) \in \mathbb{R}^{3\times 3}$$

for some (unknown) scalar factor λ.

Lemma 3.3 (Normalization of the planar homography)**.** For a matrix of the form $H_L = \lambda(R + \frac{1}{d} T N^T)$, we have

$$|\lambda| = \sigma_2(H_L),$$

where $\sigma_2(H_L) \in \mathbb{R}$ is the second largest singular value of H_L.

Then, if $\{\sigma_1, \sigma_2, \sigma_3\}$ are the singular values of H_L recovered from linear least-squares estimation, we set a new

$$H = H_L / \sigma_2(H_L).$$

Thid recovers H up to the form $H = \pm(R + \frac{1}{d} T N^T)$. To get the correct sign, we may use $\lambda_2^j \mathbf{x}_2^j = H \lambda_1^j \mathbf{x}_1^j$ and the fact that $\lambda_1^j, \lambda_2^j > 0$ to impose the positive depth constraint

$$(\mathbf{x}_2^j)^T H \mathbf{x}_1^j > 0, \forall j = 1, 2, \ldots, n.$$

3.4.3 Decomposing the planar homography matrix

Theorem 3.6 (Decomposition of the planar homography matrix)**.** Given a matrix $H = (R + \frac{1}{d} T N^T)$, there are at most two physically possible solutions for a decomposition into parameters $\{R, \frac{1}{d}T, N\}$ given below.

$$\begin{cases} R_1 = W_1 U_1^T \\ N_1 = \widehat{v_2} u_1 \\ \frac{1}{d} T_1 = (H - R_1) N_1 \end{cases} \qquad \begin{cases} R_2 = W_2 U_2^T \\ N_2 = \widehat{v_2} u_2 \\ \frac{1}{d} T_2 = (H - R_2) N_2 \end{cases}$$

$$\begin{cases} R_3 = W_1 U_1^T = R_1 \\ N_1 = -\widehat{v_2} u_1 = -N_1 \\ \frac{1}{d} T_3 = -(H - R_1) N_1 = -\frac{1}{d} T_1 \end{cases} \qquad \begin{cases} R_4 = W_2 U_2^T = R_2 \\ N_4 = -\widehat{v_2} u_2 = -N_2 \\ \frac{1}{d} T_4 = -(H - R_2) N_2 = -\frac{1}{d} T_2 \end{cases}$$

Algorithm 4: The four-point algorithm for a planar scene

Function ():

Input:

Output:

For a given set of image pairs $(\mathbf{x}_1^j, \mathbf{x}_2^j), j = 1, 2, \ldots, n (n \geq 4)$, of points on a plane $N^T\mathbf{X} = d$, this algorithm finds $\{R, \frac{1}{d}T, N\}$ that solves

$$\widehat{\mathbf{x}_2^j}^T \left(R + \frac{1}{d}TN^T\right)\mathbf{x}_1^j = 0, j = 1, 2, \ldots n.$$

(a) Compute a first approximation of the homography matrix. Construct $\chi = \begin{bmatrix}\mathbf{a}^1 & \mathbf{a}^2 & \cdots \mathbf{a}^n\end{bmatrix}^T \in \mathbb{R}^{3n\times 9}$ from correspondences $\mathbf{x}_1^j$ and $\mathbf{x}_2^j$, where $\mathbf{a}^j = \mathbf{x}_1^j \otimes \widehat{\mathbf{x}_2^j} \in \mathbb{R}^{9\times 3}$. Find the vector $H_L^s \in \mathbb{R}^9$ of unit length that solves column of V_χ. Unstack the nine elements of H_L^s into a square 3×3 matrix H_L.

(b) Normalization of the homography matrix. Compute the singular values $\{\sigma_1, \sigma_2, \sigma_3\}$ of the matrix H_L and normalize it as

$$H = H_L/\sigma_2.$$

Correct the sign of H according to sign $(\mathbf{x}_2^j)^T H \mathbf{x}_1^j$ for $j = 1, 2, \ldots, n$.

(c) Decomposition of the homography matrix. Compute the singular value decomposition of

$$H^T H = V\Sigma V^T.$$

and compute the four solutions for a decomposition $\{R, \frac{1}{d}T, N\}$. Select the two physically possible ones by imposing the positive depth constraint $N^T\mathbf{e}_3 > 0$.

end

3.4.4 Relationships between the homography and the essential matrix

In practice, when the scene is piecewise planar, we may need to compute the essential matrix E with the given homography H or compute the homography using the essential matrix E which is estimated using points in general position.

Theorem 3.7 (Relationshipps between the homography and essential matrix)**.** For a matrix $E = \widehat{T}R$ and a matrix $H = R + Tu^T$ for some nonsingular $R \in \mathbb{R}^{3\times 3}, T, u \in \mathbb{R}^3$, with $\|T\| = 1$, we have:

(a) $E = \widehat{T}H$;
(b) $H^T E + E^T H = 0$;
(c) $H = \widehat{T}^T E + Tv^T$, for some $v \in \mathbb{R}^3$.

Corollary 3.1 (From homography to the essential matrix)**.** Given a homography H and two pairs of images $(\mathbf{x}_1^i, \mathbf{x}_2^i), i = 1, 2$ of two points not on the plane P from which H is induced, we have

$$E = \widehat{T}H,$$

where $T \sim \widehat{\ell_1^1}\ell_2^2$ and $\|T\| = 1$.

Corollary 3.2 (From essential matrix to homography)**.** Given an essential matrix E and three pairs of images $(\mathbf{x}_1^i, \mathbf{x}_2^i), i = 1, 2, 3$, of three points in 3-D, the homography H induced by the plane specified by the three points then is

$$H = \widehat{T}^T E + T v^T,$$

where $v = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}^T \in \mathbb{R}^3$ solves the system of three linear equations

$$\widehat{\mathbf{x}_2^i}(\widehat{T}^T E + T v^T)\mathbf{x}_1^i = 0, i = 1, 2, 3.$$

3.5 Continuous motion case

3.5.1 Continuous epipolar constraint and the continuous essential matrix

Let us assume that camera motion is described by a smooth (i.e., continuously differential) trajectory $g(t) = (R(t), T(t)) \in SE(3)$ with body velocities $(\omega(t), v(t)) \in se(3)$. For a point $p \in \mathbb{R}^3$, its coordinates as a function of time $\mathbf{X}(t)$ satisfy

$$\dot{\mathbf{X}}(t) = \widehat{\omega}(t)\mathbf{X}(t) + v(t).$$

The image of the point P taken by the camera is the vector $\mathbf{x}$ that satisfies $\lambda(t)\mathbf{x}(t) = \mathbf{X}(t)$. Denote the velocity of the image point $\mathbf{x}$ by $\mathbf{u} \doteq \dot{\mathbf{x}} \in \mathbb{R}^3$. The velocity $\mathbf{u}$ is also called *image motion field*, which under the brightness constancy assumption can be approximated by the *optical flow*. To obtain an explicit expression for $\mathbf{u}$, we notice that

$$\mathbf{X} = \lambda\mathbf{x}, \dot{\mathbf{X}} = \dot{\lambda}\mathbf{x} + \lambda\dot{\mathbf{x}}.$$

Then

$$\dot{\mathbf{x}} = \widehat{\omega}\mathbf{x} + \frac{1}{\lambda}v - \frac{\dot{\lambda}}{\lambda}\mathbf{x}.$$

Then the image velocity $\mathbf{u} = \dot{\mathbf{x}}$ depends not only on the camera motion but also on the depth scale λ of the point. For the planar perspective projection and the spherical perspective projection, the expression for $\mathbf{u}$ will be slightly different. To eliminate the depth scale λ, consider now the inner product of the vectors with the vector $v \times \mathbf{x}$. We obtain

$$\dot{\mathbf{x}}^T\widehat{v}\mathbf{x} = \mathbf{x}^T\widehat{\omega}^T\widehat{v}\mathbf{x}.$$

We can rewrite the above equation in an equivalent way:

$$\mathbf{u}^T\widehat{v}\mathbf{x} + \mathbf{x}^T\widehat{\omega}\widehat{v}\mathbf{x} = 0,$$

which is called the *continuous epipolar constraint*.

Lemma 3.4. Consider the matrices $M_1, M_2 \in \mathbb{R}^{3\times3}$. Then $\mathbf{x}^T M_1\mathbf{x} = \mathbf{x}^T M_2\mathbf{x}$ for all $\mathbf{x} \in \mathbb{R}^3$ if and only if $M_1 - M_2$ is a skew-symmetric matrix, i.e. $M_1 - M_2 \in so(3)$.

4 Convolutional Neural Networks for Visual Recognition

This portion is from Stanford CS231n.

4.1 Image Classification

The Image Classification problem is the task of assigning an input image one label from a fixed set of categories. Many other seemingly distinct Computer Vision tasks (such as object detection, segmentation) can be reduced to image classification.

4.1.1 Challenges

Since this task of recognizing a visual concept (e.g. cat) is relatively trivial for a human to perform, it is worth considering the challenges involved from the perspective of a Computer Vision algorithm. It is known that the raw representation of images as a 3-D array of brightness values:

- Viewpoint variation. A single instance of an object can be oriented in many ways with respect to the camera.
- Scale variation. Visual classes often exhibit variation in their size (size in the real world, not only in terms of their extent in the image).
- Deformation. Many objects of interest are not rigid bodies and can be deformed in extreme ways.
- Occlusion. The objects of interest can be occluded. Sometimes only a small portion of an object (as little as few pixels) could be visible.
- Illumination conditions. The effects of illumination are drastic on the pixel level.
- Background clutter. The objects of interest may blend into their environment, making them hard to identify.
- Intra-class variation. The classes of interest can often be relatively broad, such as *chair*. There are many different types of these objects, each with their own appearance.

4.1.2 Data-driven approach

Unlike writing an algorithm for, for example, sorting a list of numbers, it is not obvious how one might write an algorithm for identifying cats in images. Therefore, instead of trying to specify what every one of the categories of interest look like directly in code, the approach that we will take is not unlike one you would take with a child: we're going to provide the computer with many examples of each class and then develop learning algorithms that look at these examples and learn about the visual appearance of each class. This approach is referred to a *data-driven approach*, since it relies on first accumulating a *training dataset* of labeled images.

4.1.3 The image classification pipeline

The task in Image Classification is to take an array of pixels that represents a single image and assign a label to it. The complete pipeline can be formalized as follows:

- Input. Out input consists of a set of N images, each labeled with one of K different classes. We refer to this data as the *training data*.
- Learning. Our task is to use the training set to learn what every one of the classes looks like. We refer to this step as *training a classifier*, or *learning a model*.
- Evaluation. In the end, we evaluate the quality of the classifier by asking it to predict labels for a new set of images that it has never seen before. We will then compare the true labels of these images to the ones predicted by the classifier. Intuitively, we're hoping that a lot of the predictions match up with the true answers (which we call the *ground truth*).

4.2 Nearest Neighbor Classifier

This classifier has nothing to do with Convolutional Neural Networks and it is very rarely used in practice. Suppose now that we are given the CIFAR-10 training set of $50,000$ images ($5,000$ images for every one of the labels), and we wish to label the remaining $10,000$. The nearest neighbor classifier will take a test image, compare it to every single one of the training images, and predict the label of the closest training image. The images are blocks of $32 \times 32 \times 3$. One of the simplest possibilities is to compare the images pixel by pixel and add up all the differences. In other words, given two images and representing them as vectors I_1, I_2, a reasonable choice for comparing them might be the L_1 distance:

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|.$$

We can also use the L_2 distance:

$$d_2(I_1, I_2) = \left[\sum_p (I_1^p - I_2^p)^2\right]^{1/2}.$$

Note that both L_1 and L_2 distance are special cases of the p-norm:

$$d_p(I_1, I_2) = \left[\sum_q |I_1^q - I_2^q|^p\right]^{1/p}.$$

4.3 k-Nearest Neighbor Classifier

Instead of finding the single closest image in the training set, we will find the top k closest images, and have them vote on the label of the test image. in particular, when $k = 1$, we recover the Nearest Neighbor classifier. Intuitively, higher values of k have a smoothing effect that makes the classifier more resistant to outliers.

4.4 Validation sets for hyper-parameter tuning

The k-nearest neighbor classifier requires a setting for k. But what number works best? Additionally, we saw that there are many different distance functions we could have used: L_1 norm, L_2 norm, dot products, and etc. These choices are called *hyper-parameters*. We should try out many different values and see what works best, but we cannot use the test set for the purpose of tweaking hyper-parameters. The test set should never touched until one time at the very end. Otherwise, the tuned hyper-parameters only work well on the test set (overfitting) but the performance is significantly reduced when the model is deployed. The ideal way to tune the hyper-parameters is to split our training set in two: a slightly smaller training set, and a *validation set*. This validation set is essentially used as a fake test set to tune the hyper-parameters.

4.4.1 Cross-validation

In cases where the size of the training data is small, a more sophisticated technique for hyper-parameter tuning called *cross-validation* is used. We can split the training data into n equal folds, use $n - 1$ folds of them for training, and 1 for validation. We would then iterate over which fold is the validation fold, evaluate the performance, and finally average the performance across the different folds.

4.4.2 In practice

In practice, people prefer to avoid cross-validation in favor of having a single validation split, since cross-validation can be computationally expensive. The splits people tend to use is between $50\% - 90\%$ of the training data for training and rest for validation. If the number of hyper-parameters is large, you may prefer to use bigger validation splits. If the number of examples in the validation set is small, it is safer to use cross-validation. Typical number of folds in practice would be 3-fold, 5-fold, or 10-fold cross-validation.

4.5 Linear Classification

4.5.1 Overview

An approach to image classification that will be extended to entire Neural Networks and Convolutional Neural Networks will be developed. The approach has two major components: a *score function* that maps the raw data to class scores, and a *loss function* that quantifies the agreement between the predicted scores and the ground truth labels. Then we cast this as an optimization problem in which we will minimize the loss function with respect to the parameters of the score function.

4.5.2 Parameterized mapping from images to label scores

Assume that a training dataset of images $\mathbf{x}_i \in \mathbb{R}^D$, each associated with a label $y_i, i = 1, \ldots, N$ and $y_i \in \{1, \ldots, K\}$. That is, we have N examples, (each with a dimensionality D) and K distinct categories. Define the score function $f : \mathbb{R}^D \to \mathbb{R}^K$ that maps the pixel values of an image to confidence scores for each class.

4.5.3 Linear classifier

Define $f : \mathbb{R}^D \to \mathbb{R}^K$ as the following linear mapping:

$$f(\mathbf{x}_i, W, \mathbf{b}) = W\mathbf{x}_i + \mathbf{b}.$$

where the image $\mathbf{x}_i \in \mathbb{R}^{D \times 1}$ has all of its pixels flattened out to a single column vector of shape $D \times 1$, the weights $W \in \mathbb{R}^{K \times D}$, and the bias $\mathbf{b} \in \mathbb{R}^{K \times 1}$. Our goal is to find W and $\mathbf{b}$ such that computed scores match the ground truth labels across the whole training set.

4.5.4 Interpreting a linear classifier

Note that a linear classifier computes the score of a class as a weighted sum of all of its pixel values across all 3 of its color channels. Depending on precisely what values we set for these weights, the function has the capacity to like or dislike (depending on the sign of each weight) certain colors at certain positions in the image.

4.5.5 Analogy of images as high-dimensional points

Since the images are stretched into high-dimensional column vectors, we can interpret each image as a single point in this space. Analogously, the entire dataset is a (labeled) set of points. As we saw above, each row of W is a classifier for one of the classes. The geometric interpretation of these numbers is that as we change one of the rows of W, the corresponding line in the pixel space will rotate in different directions. The biases $\mathbf{b}$, on the other hand, allow our classifiers to translate the

lines. In particular, note that without the bias terms, plugging in $\mathbf{x}_i = \mathbf{0}$ would always give score of zero regardless of the weights, so all lines would be forces to cross the origin.

4.5.6 Interpretation of linear classifier as template matching

Another interpretation for the weights W is that each row of W corresponds to a *template* (*prototype*) for one of the classes. The score of each class for an image is then obtained by comparing each template with the image using an *inner product*one by one to find the one that "fits" best. With this terminology, the linear classifier is doing template matching, where the templates are learned. Another way to think of it is that we are still effectively doing Nearest Neighbor, but instead of having thousands of training images we are only using a single image per class (although we will learn it, and it does not necessarily have to be one of the images in the training set), and we use the (negative) inner product as the distance instead of the L_1 or L_2 distance.
The linear classifier is to weak to properly account for different-colored cars, a neural network will be able to develop intermediate neurons in its hidden layers that could detect specific car types, and neurons on the next layer could combine these into a more accurate car score through a weighted sum of the individual car vectors.

4.5.7 Bias trick

We can represent the two parameters $W, \mathbf{b}$ as one. A commonly used trick is to combine the two sets of parameters into a single matrix that holds both of them by extending the vector $\mathbf{x}_i$ with one additional dimension that always hold the constant 1 - a default *bias dimension*. That is,

$$W' = \begin{bmatrix} W & \mathbf{b} \end{bmatrix} \in \mathbb{R}^{K \times (D+1)}, \mathbf{x}'_i = \begin{bmatrix} \mathbf{x}_i \\ 1 \end{bmatrix} \in \mathbb{R}^{(D+1) \times 1}.$$

Then

$$f(\mathbf{x}_i, W, \mathbf{b}) = W\mathbf{x}_i + \mathbf{b} = \begin{bmatrix} W & \mathbf{b} \end{bmatrix} \begin{bmatrix} \mathbf{x}_i \\ 1 \end{bmatrix} = W' + \mathbf{x}'_i = f'(\mathbf{x}'_i, W'),$$

where $f : \mathbb{R}^D \to \mathbb{R}^K$ and $f' : \mathbb{R}^{D+1} \to \mathbb{R}^K$.

4.5.8 Image data preprocessing

In Machine Learning, it is a very common practice to always perform normalization of the input features (every pixel is thought of as a feature). In particular, it is important to *center the data* by subtracting the mean from every feature. Further common preprocessing is to scale each input feature so that its values range from $[-1, 1]$.

4.6 Loss function

We are going to measure our unhappiness with outcomes with a *loss function* (*cost function*, or the *objective function*). Intuitively, the loss will be high if we're doing a poor job of classifying the training data, and it will be low if we're doing well.

4.6.1 Multi-class Support Vector Machine (SVM) loss

The SVM loss is set up so that the SVM "wants" the correct class for each image to a have a score higher than the incorrect classes by some fixed margin Δ. For the ith example, we are given the pixels of image $\mathbf{x}_i$ and the label y_i that specifies the index of the correct class. The score function

takes the pixels and computes the vector $\mathbf{s} = f'(\mathbf{x}'_i, W')$ of class scores. For example, the jth score for the jth class is the jth element $\mathbf{s}_j = f'(\mathbf{x}'_i, W')_j$. The Multi-class SVM loss for the ith example is then formalized as follows:

$$\begin{aligned} L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta) \\ &= \sum_{j \neq y_i} \max(0, f'(\mathbf{x}'_i, W')_j - f'(\mathbf{x}'_i, W')_{y_i} + \Delta) \\ &= \sum_{j \neq y_i} \max(0, \mathbf{w}_j^T \mathbf{x}'_i - \mathbf{w}_{y_i}^T \mathbf{x}'_i + \Delta), \end{aligned}$$

where $\mathbf{w}_j$ is the jth row of W' reshaped as a column. In summary, the SVM loss function wants the score of the correct class y_i to be larger than the incorrect class scores by at least Δ. If this is not the case, we will accumulate the loss. Note that the threshold at zero $\max(0, \cdot)$ function is often called the *hinge loss*. Also, we have *squared hinge loss* which has the form $\max(0, \cdot)^2$. Setting Δ. It turns out that Δ can safely be set to $\Delta = 1.0$ in all cases. The hyper-parameters Δ and λ, in fact, both control the same trade-off: The trade-off between the data loss and the regularization loss in the objective. The key to understanding this is that the magnitude of the weights W has direct effect on the scores (and hence also their differences): As we shrink all values inside W the score differences will becomes lower, and as we scale up the weights the score differences will all become higher. Therefore, the exact value of the margin between the scores, e.g. $\Delta = 1$ or $\Delta = 100$) is in some sense meaningless because the weights can shrink or stretch the differences arbitrarily.

4.6.2 Regularization

The W we obtained from the loss function is not necessarily unique: there might be many similar W that correctly classify the examples, i.e., $\lambda W, \lambda > 1 \in \mathbb{R}$. We wish to encode some preference for a certain set of weights W over others to remove this ambiguity. We can do so by extending the loss function with a regularization term $R(W)$. The most common regularization penalty is the L_2 norm that discourages large weighs through an element-wise quadratic penalty over all parameters:

$$R(W) = \sum_k \sum_l W_{k,l}^2.$$

Including the regularization penalty completes the full Multi-class Support Vector Machine loss, which is made up of two components: the *data loss* (which is the average loss L_i over all examples) and the *regularization loss*. That is, the full Multi-class SVM loss becomes:

$$L = \frac{1}{N} \sum_i L_i + \lambda R(W) = \frac{1}{N} \sum_i \sum_{j \neq y_i} \max(0, f'(\mathbf{x}'_i, W')_j - f'(\mathbf{x}'_i, W')_{y_i} + \Delta) + \lambda \sum_k \sum_l W_{k,l}^2,$$

where N is the number of examples.

4.7 Softmax classifier

The *Softmax classifier* is the generalization of the binary Logistic Regression classifier to multiple classes. Unlike the SVM which treats the outputs $f'(\mathbf{x}'_i, W')$ as (uncalibrated and possibly difficult to interpret) scores for each class, the Softmax classifier gives a slightly more intuitive output (normalized class probabilities) and also has a probabilistic interpretation that we will describe

shortly. In the Softmax classifier, the function mapping $f'(\mathbf{x}'_i, W') = W'\mathbf{x}'_i$ stays unchanged, but we now interpret these scores as the unnormalized log probabilities for each class and replace the *hinge loss* with a *cross-entropy loss* that has the form:

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) = -\log e^{f_{y_i}} + \log\sum_j e^{f_j} = -f_{y_i} + \log\sum_j e^{f_j},$$

where f_j is the jth element of the vector of class scores f, the function $s_i(f) = e^{f_{y_i}}/\sum_j e^{f_j}$ is the softmax function. The full loss function now becomes

$$L = \frac{1}{N}\sum_i L_i + \lambda R(W) = \frac{1}{N}\sum_i\left(-f_{y_i} + \log\sum_j e^{f_j}\right) + \lambda\sum_k\sum_l W_{k,l}^2.$$

4.7.1 Cross-entropy (Information theory view)

The *cross-entropy* between a "true" distribution p and an estimated distribution q is defined as

$$H(p,q) = -\sum_{i=1}^{K} p_i(f)\log q_i(f).$$

The Softmax classifier is hence minimizing the cross-entropy between the estimated class probabilities $q_i(f) = e^{f_{y_i}}/\sum_j e^{f_j}$ and the "true" distribution, which in this interpretation is the distribution where all probability mass is on the correct class (i.e., $p = [0,\ldots,0,1,0,\ldots,0]$ contains a single 1 at the y_i-th position.) Moreover, since the cross-entropy can be written in terms of entropy and the Kullback-Leibler divergence as $H(p,q) = H(p) + D_{KL}(p||q)$, and the entropy of the delta function p is zero, this is also equivalent to minimizing the KL divergence between the two distributions (a measure of distance). In other words, the cross entropy objective *wants* the predicted distribution to have all of its mass on the correct answer.

4.7.2 Probabilistic interpretation

The expression

$$\begin{aligned} s_i(f) &= P(y_i|\mathbf{x}', W') \\ &= \frac{P(y_i, \mathbf{x}', W')}{P(\mathbf{x}', W')} \\ &= \frac{P(y_i, \mathbf{x}', W')}{\sum_{i=1}^{K} P(y_i, \mathbf{x}', W')} \\ &= \frac{e^{f_{y_i}(\mathbf{x}', W')}}{\sum_{i=1}^{K} e^{f_{y_i}(\mathbf{x}', W')}} \\ &= \frac{e^{f_{y_i}(\mathbf{x}', W')}}{\sum_{j=1}^{K} e^{f_j(\mathbf{x}', W')}} \end{aligned}$$

can be interpreted as the (normalized) probability assigned to the correct label y_i given the image $\mathbf{x}_i$ and parameterized by W. In probabilistic interpretation, we are therefore minimizing the negative log likelihood of the correct class, which can be interpreted as performing *Maximum Likelihood Estimation* (MLE). A nice feature of this view is that we can now also interpret the regularization term $R(W)$ in the full loss function as coming from a Gaussian prior over the weight matrix W, where instead of MLE we are performing the *Maximum a posteriori* (MAP) estimation.

4.7.3 Practical issues: numerical stability

When computing the Softmax function , the intermediate terms $e^{f_{y_i}}$ and $\sum_j e^{f_j}$ may be very large due to the exponentials. Dividing large numbers can be numerically unstable, so it is important to use a normalization trick.

$$\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} = \frac{Ce^{f_{y_i}}}{C\sum_j e^{f_j}} = \frac{e^{f_{y_i}+\log C}}{\sum_j e^{f_j+\log C}},$$

where C can be chosen to be $\log C = -\max_j f_j$.

4.8 Optimization

Optimization is the process of finding the set of parameters W that minimize the loss function. To reiterate, the loss function lets us quantify the quality of any particular set of weights W. Here are the strategies:

(a) Random search. Since it is so simple to check how good a given set of parameters W is, we may simply try out many different random weights and keep track of what works best. The *core idea* is that finding the best set of weights W is a very difficult or even impossible problem (especially once W contains weights for entire complex neural networks), but the problem of refining a specific set of weights W to be slightly better is significantly less difficult. In other words, our approach will be to start with a random W and then iteratively refine it, making it slightly better each time.

(b) Random Local Search. We will start out with a random W, generate random perturbations δW to it and if the loss at the perturbed $W + \delta W$ is lower, we will perform an update.

(c) Following the Gradient. We can compute the *best* direction along which we should change our weight vector that is mathematically guaranteed to be the direction of the steepest descend (at least in the limit as the step size goes towards zero). This direction will be related to the *gradient* of the loss function. The gradient is a generalization of slope for functions that take a vector of numbers.

4.8.1 Computing the gradient numerically with finite differences

The derivative of a function f at a point x can be approximated using finite difference as follows,

$$f'(x) \approx \begin{cases} \dfrac{f(x+h)-f(x)}{h}, & \text{forward difference} \\ \dfrac{f(x)-f(x-h)}{h}, & \text{backward difference} \\ \dfrac{f(x+h)-f(x-h)}{2h}. & \text{central difference} \end{cases}$$

Ideally, you want to use the smallest step size that does not lead to numerical issues. Additionally, the central difference works better. The gradient tells us the direction in which the function has the steepest rate of increase, but it does not tell us how far along this direction we should step. Choosing the step size (*learning rate*) will become one of the most important hyper-parameter settings in training a neural network. However, this strategy is not scalable when the neural networks reaches tens of millions of parameters.

4.8.2 Computing the gradient analytically with Calculus

Unlike the numerical gradient, the analytic gradient can be more error prone to implement, which is hwy in practice it is very common to compute the analytic gradient and compare it to the numerical gradient to heck the correctness of the implementation. This is called a *gradient check*.
Take the SVM loss function for a single data point for example:

$$L_i = \sum_{j \neq y_i} \max(0, \mathbf{w}_j^T \mathbf{x}_i' - \mathbf{w}_{y_i}^T \mathbf{x}_i' + \Delta).$$

We can differentiate L_i with respect to $\mathbf{w}_{y_i}$,

$$\nabla_{\mathbf{w}_{y_i}} L_i = -\left[\sum_{j \neq y_i} \mathbf{1}(\mathbf{w}_j^T \mathbf{x}_i' - \mathbf{w}_{y_i}^T \mathbf{x}_i' + \Delta > 0)\right] \mathbf{x}_i',$$

where $\mathbf{1}$ is the indicator function. For other rows where $j \neq y_i$, the gradient is

$$\nabla_{\mathbf{w}_j} L_i = \mathbf{1}(\mathbf{w}_j^T \mathbf{x}_i' - \mathbf{w}_{y_i}^T \mathbf{x}_i' + \Delta > 0)\mathbf{x}_i'.$$

In large-scale applications, the training data can have on order of millions of examples. Hence, it seems wasteful to compute the full lose function over the entire training set in order to perform only a single parameter update. A very common approach to addressing this challenge is to compute the gradient over *batches* of the training data. For example, in current state of the art ConvNets, a typical batch contains 256 examples from the entire set of 1.2 million. The reason this works well is that the examples in the training data are correlated. The extreme case of this is a setting where the mini-batch contains only a single example. The process is called *Stochastic Gradient Descent* (SGD) (also sometimes *on-line* gradient descent). This is relatively less common to see because in practice due to vectorized code optimizations it can be computationally much more efficient to evaluate the gradient for 100 examples, than the gradient for one example 100 times. Even though SGD technically refers to using a single example at a time to evaluate the gradient, the term SGD is also used to refer to Mini-batch Gradient Descent (MGD), or Batch gradient descent (BGD). The size of the mini-batch is a hyper-parameter but it is not very common to cross-validate it. It is usually based on memory constraints (if any), or set to some value, say powers of 2, because many vectorized operation implementations work faster.

4.9 Backpropagation

4.9.1 Introduction

Backpropagation is a way of computing gradients of expressions through recursive application of chain rule. Understanding of this process and its subtleties is critical to understand, and effectively develop, design and debug Neural Networks. Given a function $f(\mathbf{x})$ where $\mathbf{x}$ is a vector of inputs and we are interested in computing the gradient of f at $\mathbf{x}$, i.e.,

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_N} \end{bmatrix}^T, \mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_N \end{bmatrix}.$$

The motivation is that in the specific case of Neural Networks, f will correspond to the loss function L and the inputs $\mathbf{x}$ will consist of the training data and the neural network weights. For example, the loss could be the SVM loss function and the inputs are both the training data $\mathbf{x}_i, y_i, i = 1, 2, \ldots, N$ and the weights W and bias $\mathbf{b}$. Note that (as is usually the case in Machine Learning) we think of

the training data as given and fixed, and of the weights as variables we have control over. Hence, even though we can easily use backpropagation to compute the gradient on the input examples $\mathbf{x}_i$., in practice we usually only compute the gradient for the parameters so that we can use it to perform a parameter update. However, the gradient on $\mathbf{x}_i$ can still be useful sometimes, for example for purposes of visualization and interpreting what the Neural Network might be doing.

4.9.2 Compound expressions with chain rule

Assume that $f(x, y, z) = (x + y)z$. Let $q = x + y$, then $f(x, y, z) = qz$. It is known that

$$\begin{aligned}\frac{\partial f}{\partial x} &= \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = z \cdot 1 = z,\\ \frac{\partial f}{\partial y} &= \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = z \cdot 1 = z,\\ \frac{\partial f}{\partial z} &= \frac{\partial f}{\partial z}\frac{\partial z}{\partial z} = q \cdot 1 = x + y.\end{aligned}$$

The real-valued "circuit" on left shows the visual representation of the computation. The *forward pass* computes values from inputs to output. The *backward pass* then performs backpropagation which starts at the end and recursively applies the chain rule to compute the gradients all the way to the inputs of the circuit. The backpropagation is a beautifully local process. Every gate in a circuit diagram gets some inputs and can right away compute its output value and the *local* gradient of its inputs with respect to its output value.

4.9.3 Modularity: Sigmoid example

Any kind of differentiable function can act as a gate, and we can group multiple gates into a single gate, or decompose a function into multiple gates whenever it is convenient. To illustrate this, consider the *sigmoid activation function*

$$f(\mathbf{w}, \mathbf{x}) = \frac{1}{1 + e^{-w_0x_0+w_1x_1+w_2}},$$

which describes a 2-dimensional neuron (with inputs $\mathbf{x}$ and weights $\mathbf{w}$). Let

$$\begin{aligned}p_0(x) = -w_0x &\implies p_0'(x) = -w_0,\\ p_1(x) = w_1x &\implies p_1'(x) = w_1,\\ q(x) = x + w_2 &\implies q'(x) = 1,\\ r(x) = e^{-x} &\implies r'(x) = -e^{-x} = -r(x),\\ s(x) = 1 + x &\implies s'(x) = 1,\\ t(x) = \frac{1}{x} &\implies = t'(x) = -\frac{1}{x^2}.\end{aligned}$$

Then

$$f(\mathbf{w}, \mathbf{x}) = t(s(r(q(p_0(x_0) + p_1(x_1))))) \implies \frac{\partial f}{\partial x_0} = \frac{\partial f}{\partial t}\frac{\partial t}{\partial s}\frac{\partial s}{\partial r}\frac{\partial r}{\partial q}\frac{\partial q}{\partial p_0}\frac{\partial p_0}{\partial x_0}.$$

These operations is implementing the *sigmoid function* $\sigma(x)$,

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

Then taking the derivative with respect to x gives

$$\frac{d\sigma}{dx}(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1+e^{-x}-1}{1+e^{-x}}\frac{1}{1+e^{-x}} = [1-\sigma(x)]\sigma(x).$$

Therefore, in any real practical application it would be very useful to group these operations into a single gate.

4.9.4 Backpropagation in practice: staged computation

(a) Cache forward pass variables. To compute the backward pass it is very helpful to have some of the variables that were used in the forward pass. In practice we want to cache these variables, and so that they are available during propagation.
(b) Gradients add up at forks. Following the *multivariate chain rule*, which states that if a variable branches out to different parts of the circuit, then the gradient that flow back to it will add.
(c) The *add gate* always takes the gradient on its output and distributes it equally to all of its inputs, regardless of what their values were during the forward pass.
(d) The *max gate* routes the gradient.
(e) The *multiply gate*'s local gradients are the input values (except switched), and this is multiplied by the gradient on its output during the chain rule.

Further reading: Matrix/vector derivatives by Erik Learned-Miller, Automatic differentiation in machine learning: a survey.

4.10 Convolutional Neural Networks (CNNs/ConvNets)

Convolutional Neural Networks are very similar to ordinary Neural Networks: they are made up of neurons that have learnable weights and biases. Each neuron receives some input, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class cores at the other. And they still have a loss function (e.g., SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks we developed for learning regular Neural Networks still apply.

4.10.1 Architecture Overview

(a) Regular Neural Nets. Neural Networks receive an input (a single vector), and transform it through a series of *hidden layers*. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. They last fully-connected layer is called the "output layer" and in classification settings it represents the class scores. However, Regular Neural Nets don't scale well to full images. The full connectivity is wasteful and the huge number of parameters would quickly lead to over-fitting.
(b) 3D volumes of neurons. Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. The neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, by the end of the ConvNet architecture we will reduce the full image into a single vector of class scores, arranged along the depth dimension. In a word, a ConvNet is made up of layers. Every layer has a simple

API: it transforms an input 3D volume to an output 3D volume with some differentiable function that may or may not have parameters.

4.10.2 Layers used to build ConvNets

A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures. *Convolutional Layer* (CONV), *Pooling Layer* (POOL), and *Fully-Connected Layer* (FC). Stacking these layers forms a full ConvNet architecture: [INPUT - CONV - RELU - POOL - FC].

- INPUT will hold the raw pixel values of the image with three color channels R, G, B.
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. The depth of the output volume is the number of filters being used.
- RELU layer will apply an element-wise activation function, such as the $\max(0, x)$ thresholding at zero. This leaves the size of the volume unchanged.
- POOL layer will perform a down-sampling operation along the spatial dimensions (width, height).
- FC layer will compute the class scores, resulting in volume of size, where each of the 10 numbers correspond to a class score.

In this way, ConvNets transform the original image layer by layer from the original pixel values to the final class scores. Note that some layers contain parameters and other don't. In particular, the CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons). On the other hand, the RELU/POOL layers will implement a fixed function. The parameters in the CONV/FC layers will be trained with gradient descent so that the class scores that the Convnet computes are consistent with the labels in the training set for each image.

4.10.3 Convolutional Layer

The CONV layer is the core building block of a Convolutional Network that does most of the computational heavy lifting.

- Overview and intuition. The CONV layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a ConvNet might have size $5 \times 5 \times 3$ (i.e., 5 pixels width and height, and 3 because images have depth 3, the color channels). During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. Now, will have an entire set of filters in each CONV layer, and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.

- Local connectivity. When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume. Instead, we will connect each neuron to only a local region of the input volume. The spatial extent of this connectivity is a hyper-parameter called the *receptive field* of the neuron (equivalently this is the filter size). The extent of the connectivity along the depth axis is always equal to the depth of the input volume. It is important to emphasize again this asymmetry in how we treat the spatial dimensions (width and height) and the depth dimension: The connections are local in space (along width and height), but always full along the entire depth of the input volume.
- Spatial arrangement. Three hyper-parameters control the size of the output volume: the *depth*, *stride* and *zero-padding*.
  (a) The depth of the output volume corresponds to the number of filters, each learning to look for something different in the input. For example, if the first CONV layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edges, or blobs of color. We will refer to a set of neurons that are all looking at the same region of the input as a *depth column*.
  (b) The stride is the number of pixels we slide the filter at a time. The bigger the stride, the smaller it produces the output volumes spatially.
  (c) Sometimes it will be convenient to pad the input volume with zeros around the border. The nice feature of zero-padding is that it will allow us to control the spatial size of the output volumes. For example, we can preserve the spatial size of the input volume so the input and output width and height are the same.

  The formula for calculating how many neurons "fit" is given by

$$\frac{I - F + 2P}{S} + 1,$$

  where I is the input volume size, F is the receptive field size of the CONV layer neurons, S is the stride, and P is the amount of zero padding used on the border.

  – Use of zero-padding. To ensure that the input volume and output volume will have the same size spatially, we set the zero-padding to be $P = (F - 1)/2$ when the stride $S = 1$.
  – Constraints on strides. The spatial arrangement hyper-parameters have mutual constraints.Sizing the ConvNets appropriately so that all the dimensions "work out" can be a real headache, which the use of zero-padding and some design guidelines will significantly alleviate.
- Parameter Sharing. Parameter sharing scheme is used in CONV layers to control the number of parameters. It turns out that we can dramatically reduce the number of parameters by making one reasonable assumption: That if one feature is useful to compute at some spatial position (x_1, y_1), then it should also be useful to compute at a different position (x_2, y_2). In other words, denoting a single 2-dimensional slice of depth as a *depth slice*, we are going to constrain the neurons in each depth slice to use the same weights and bias. Note that if all neurons in a single depth slice are using the same weight vector, then the forward pass of the CONV layer can in each depth slice be computed as a *convolution* of the neuron's weights with the input volume. This is why it is common to refer to the sets of weights as a *filter* (or *kernel*), that is convolved with the input.