EECS C106B / 206B Robotic Manipulation and Interaction **Spring 2020**

Lecture 12: (Image Primitives and Correspondence)

Scribes: Shiro Xu

12.1 Backgrounds

12.1.1 Motivation

Given 2 images of the same object taken from different angle (either with camera rotation or translation or both), we want to find point in both images such that they are the projection of the same 3-D point of the real object.

fig. 12.1

12.1.2 Main idea

Intuitively, human beings find image correspondence easily because not only do we look at the single point we try to match, but we look at the entire picture, compare the neighborhood, and make a decision from there.

In computer vision, we approach the correspondence problem in a very similar manner. With **Lambertian assumption**(assumes all corresponding points are the same before and after the camera motion), the general approach is to extract features from both images, compare both local and neighbor features, and find point correspondence from there.

12.2 Main challenges and Models

12.2.1 Image deformation

This challenge is due to camera's rotation and translation. In order to resolve this challenge, we first consider the case where only camera translation happened, and we use **translational model** to resolve this situation. Specifically, with d denoting the translation, we have:

$$h(x) = x + d$$

Based on translational model, we then have the **affine model** to resolve the general image deformation.

$$h(x) = Ax + d$$

12.2.2 Noise and occlusions

If 2 images are taken from different angles, then it is quite likely that an object point appears in 1 image but not the other. Using figure 12-1 as an example, the right shoulder appears in the left image but not the right, which raises a challenge of identifying such points. The model in the slide:

$$I_1(x_1) = f_o(X, g)I_2(h(x_1)) + n(h(x_1))$$

has the function f_o to account the point disappearance case. The function n at the end of the equation is to account for potential noise in the image.

12.3 Feature Extraction Methodologies

12.3.1 Optical Flow

The main idea of the optical flow is to study how image point velocity is related to the real object point velocity. For example, an application of optical flow is landing an airplane, where we need to adjust the vertical velocity of the airplane according to airplane camera in order to land smoothly. From equation 3.4 in the slide, we have:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \frac{1}{Z}\begin{bmatrix} -1 & 0 & x \\ 0 & -1 & y \end{bmatrix}\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} + \begin{bmatrix} xy & -(1+x^2) & y \\ 1+y^2 & -xy & -x \end{bmatrix}\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

where u and v are the image point velocity in x and y direction respectively. One intuitive thing from the equation is that the image point velocity is inversely proportional to Z, the depth of the real object, which is saying that the further away you are from the real object, the slower the object appears to be.

One problem emerged from the optical flow concept is **aperture problem**, as shown in the figure 12.2:

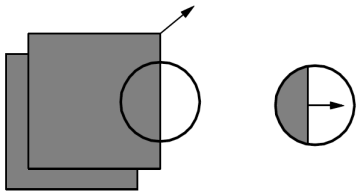

fig. 12.2, aperture problem

where if we only look at the small segment of this motion, we cannot observe the vertical component of the motion. Mathematically, if we look at the **normal flow** of the motion:

$$\mathbf{u}_n \doteq \frac{\nabla I^T \mathbf{u}}{\|\nabla I\|} \cdot \frac{\nabla I}{\|\nabla I\|} = -\frac{I_t}{\|\nabla I\|} \cdot \frac{\nabla I}{\|\nabla I\|}$$

the vertical component of the motion gets killed.

In order to generalize this kind of problems, we have the equation:

$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \mathbf{u} + \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix} = 0$$

$$G\mathbf{u} + \mathbf{b} = 0$$

and we compute the optical flow u as:

$$u = -G^{-1}b$$

Therefore, we have 3 distinct situations:
1. If G has rank 0, we can recover neither x nor y component of the velocity. An example of this is camera motion in front of a white blank wall.
2. If G has rank 1, this will be the aperture problem, and we can only recover one of the x or y component of the velocity.
3. If G has full rank, then G will be invertible and we can fully recover the velocity in both directions.

12.3.2 Convolution with Gaussian Function

Given an Gaussian function, the goal of convolving an image with an Gaussian function is to extract the high frequency component of the image and remove low frequency components (because low frequency components are mostly noise of the image). We can also adjust the standard deviation of the Gaussian function to adjust how much frequency we want to filter out. Mathematically, this is:

$$I_x[x,y] = I[x,y] * g'[x] * g[y] = \sum_{k=-\frac{w}{2}}^{\frac{w}{2}} \sum_{l=-\frac{w}{2}}^{\frac{w}{2}} I[k,l] g'[x-k] g[y-l],$$

$$I_y[x,y] = I[x,y] * g[x] * g'[y] = \sum_{k=-\frac{w}{2}}^{\frac{w}{2}} \sum_{l=-\frac{w}{2}}^{\frac{w}{2}} I[k,l] g[x-k] g'[y-l].$$

Note that because the image signal we are given is in the discrete form(pixel values), the Gaussian function above is also in the discrete form.

12.3.3 Harris Corner Detector

Given the matrix G in the section 12.3.1, the Harris Corner Detector is denoted by the formula of:

$$C(G) = det(G) + k * trace^2(G)$$

By looking at the determinant and trace, the equation is essentially looking at the size of eigenvalue of matrix G. Since the matrix G is consisted of image gradients in both x and y direction, this idea is to note any sharp change in both x and y directions(which is a property of a corner), and if so, then the detector classify such point as a corner.

12.3.4 Scale-Invariant Feature Transform

The general idea behind SIFT key is to find the dominant gradient of an image, and rotate the image such that the dominant component always point to the x=0 direction. This way, we can align images despite they are taken from different angles and orientations.

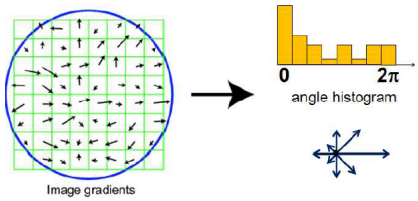

fig. 12.3, SIFT

Mathematically, the SIFT key can be denoted with the formula:

$$G(x, y, kS) = \frac{1}{2\pi(kS)^2} e^{-(x^2+y^2)/(2k^2S^2)}$$

Note that the SIFT key makes feature extraction very accurate, but it is very slow to compute. Therefore, it is relatively not frequently used if feature extraction speed is important.

12.3.5 Junction Detection and Edge Detection

In junction detection, junctions are classified into **C-Junction** and **T-Junctions**. C-Junctions are the ones that physically exist in the real object(e.g. L, W, Y points in the figure 12.4), and T-Junctions are the ones that emerge from image occlusion and may vary based on different angle(e.g. T point in the figure 12.4). Typically, T-junctions are identified by camera motion, but it is still possible to identify T-junctions from a single image if learning is incorporated in the detection.

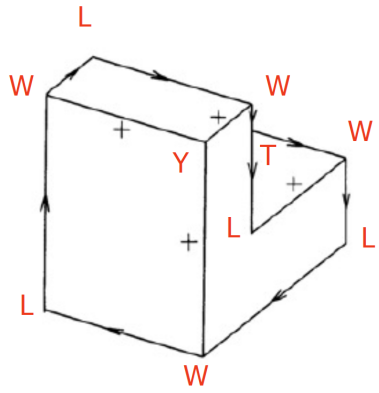

fig. 12.4, junction

What follows junction detection is edge detection, and an well-known edge detector is **Canny Edge Detector**. Essentially, we extract both the line feature and the junction feature from an image. From there, we can identify line segments.

fig. 12.4, junction

12.3.6 Learning Based Edge Detection

One intuitive feature extraction method is learning. On the one hand, learning can be incorporated into other detection methods(e.g. T-junction detection, as mentioned in 12.3.5). At the same time, it can be an extraction method on its own. With large enough data sample size, learning is proven to be a very accurate detection method. Yet, if not enough data is supplied(e.g. when the detector is seeing something new), the feature extraction can be quite inaccurate.