Two-view stereo

April 21, 2020

In prrevious lectures, we have shown that a world point $\vec{\mathbf{Q}}$ projects to an image point $\vec{\mathbf{q}}$ through the following equation:

$$\vec{\mathbf{q}} \equiv K \begin{bmatrix} R & \mathbf{t} \end{bmatrix} \vec{\mathbf{Q}} \tag{1}$$

Till now we have looked at how to estimate K, R and $\mathbf{t}$. Suppose now that we know these, or rather the camera projection matrix $P = K \begin{bmatrix} R & \mathbf{t} \end{bmatrix}$. The question is, given the image location $\vec{\mathbf{q}}$, can we find the world coordinates $\vec{\mathbf{Q}}$?

It should be clear that we do not have enough information: for a pinhole camera, a single pixel corresponds to a ray in 3D, and we don't know where along the ray the 3D point lies. However, if we have two cameras, then we can *intersect* the two resulting rays and find out the location of the 3D point (Figure 1).

What does this correspond to mathematically? Let us find out. Consider two cameras looking at the same scene, with projection matrices $\mathbf{P}^{\vec{(1)}}$ and $\mathbf{P}^{\vec{(2)}}$. Suppose a world point $\vec{\mathbf{Q}}$ projects onto the first image at $\vec{\mathbf{q}}_1$ and onto the second image at $\vec{\mathbf{q}}_2$. Then, we have the following two equations:

$$\vec{\mathbf{q}}_1 \equiv \mathbf{P}^{\vec{(1)}} \vec{\mathbf{Q}} \tag{2}$$

$$\vec{\mathbf{q}}_2 \equiv \mathbf{P}^{\vec{(2)}} \vec{\mathbf{Q}} \tag{3}$$

Let us expand these equations out. Suppose $\vec{\mathbf{q}}_1 \equiv \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$, and $\vec{\mathbf{q}}_2 \equiv \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$. Suppose $\vec{\mathbf{Q}} \equiv \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$. Then the first of these equations expands as follows:

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \equiv \begin{bmatrix} P_{11}^{(1)} & P_{12}^{(1)} & P_{13}^{(1)} & P_{14}^{(1)} \\ P_{21}^{(1)} & P_{22}^{(1)} & P_{23}^{(1)} & P_{24}^{(1)} \\ P_{31}^{(1)} & P_{32}^{(1)} & P_{33}^{(1)} & P_{34}^{(1)} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{4}$$

$$\Rightarrow \lambda \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} P_{11}^{(1)} & P_{12}^{(1)} & P_{13}^{(1)} & P_{14}^{(1)} \\ P_{21}^{(1)} & P_{22}^{(1)} & P_{23}^{(1)} & P_{24}^{(1)} \\ P_{31}^{(1)} & P_{32}^{(1)} & P_{33}^{(1)} & P_{34}^{(1)} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad \text{for some } \lambda \tag{5}$$

$$\begin{aligned} \Rightarrow \lambda x_1 &= P_{11}^{(1)} X + P_{12}^{(1)} Y + P_{13}^{(1)} Z + P_{14}^{(1)} \\ \lambda y_1 &= P_{21}^{(1)} X + P_{22}^{(1)} Y + P_{23}^{(1)} Z + P_{24}^{(1)} \\ \lambda &= P_{31}^{(1)} X + P_{32}^{(1)} Y + P_{33}^{(1)} Z + P_{34}^{(1)} \end{aligned} \tag{6}$$

Substituting the last equation in the other two, we get *two* linear equations from the first image:

$$(P_{31}^{(1)} X + P_{32}^{(1)} Y + P_{33}^{(1)} Z + P_{34}^{(1)}) x_1 = (P_{11}^{(1)} X + P_{12}^{(1)} Y + P_{13}^{(1)} Z + P_{14}^{(1)}) \tag{7}$$

$$(P_{31}^{(1)} X + P_{32}^{(1)} Y + P_{33}^{(1)} Z + P_{34}^{(1)}) y_1 = (P_{21}^{(1)} X + P_{22}^{(1)} Y + P_{23}^{(1)} Z + P_{24}^{(1)}) \tag{8}$$

$$\tag{9}$$

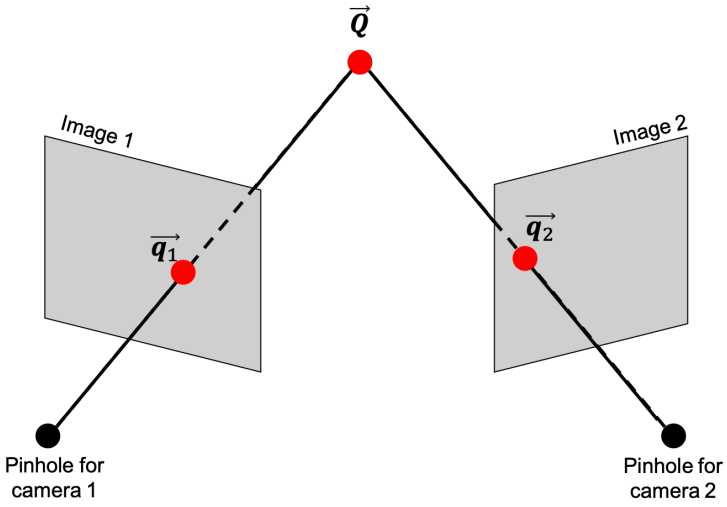

Figure 1: Setup

Because there are three variables (X, Y, Z), two equations are by themselves not enough. There is still a degree of freedom left. This is exactly equivalent to the observation that knowing the image location in one image constrains the location of a world 3D point to a line, but more information is needed to identify it exactly.

With two images, we have 4 linear equations that can be solved exactly to recover the 3D location of the world point. This is exactly equivalent to intersecting the two rays from the two cameras to find the location of the world point. This process of recovering the coordinates of the 3D world point by intersecting rays from two cameras (or solving the set of equations) is called *triangulation*.

1 Non-linear optimization

Above, we performed triangulation by solving a set of linear equations. This is a fairly straightforward approach. However, there is an alternative to this that is useful to know about because (a) it is more interpretable, and (b) it applies more generally.

The basic idea is as follows. We start from the set of equations (6). Then, instead of substituting the value of λ from the third equation to the other two, we *divide* the first two equations by the third to get:

$$x_1 = \frac{(P^{(1)}_{11}X + P^{(1)}_{12}Y + P^{(1)}_{13}Z + P^{(1)}_{14})}{(P^{(1)}_{31}X + P^{(1)}_{32}Y + P^{(1)}_{33}Z + P^{(1)}_{34})} \tag{10}$$

$$y_1 = \frac{(P^{(1)}_{21}X + P^{(1)}_{22}Y + P^{(1)}_{23}Z + P^{(1)}_{24})}{(P^{(1)}_{31}X + P^{(1)}_{32}Y + P^{(1)}_{33}Z + P^{(1)}_{34})} \tag{11}$$

This is a constraint. If (X, Y, Z) is the hypothesized 3D location of this world point, then the RHS is the *predicted* image location of this point in image 1, while the LHS is the *true* image location. Let us write the RHS as $\hat{x}_1(X, Y, Z)$ and $\hat{y}_1(X, Y, Z)$ respectively, since these are the predicted x and y coordinates in image 1, and are a function of the hypothesized X, Y, Z coordinates of the world point. Thus the constraint is that the predicted image location should match the true image location:

$$x_1 = \hat{x}_1(X, Y, Z) \qquad y_1 = \hat{y}_1(X, Y, Z) \tag{12}$$

To estimate how far off we are from satisfying this constraint, we can write down an error:

$$\Delta_1(X, Y, Z) = \sqrt{(x_1 - \hat{x}_1(X, Y, Z))^2 + (y_1 - \hat{y}_1(X, Y, Z))^2} \tag{13}$$

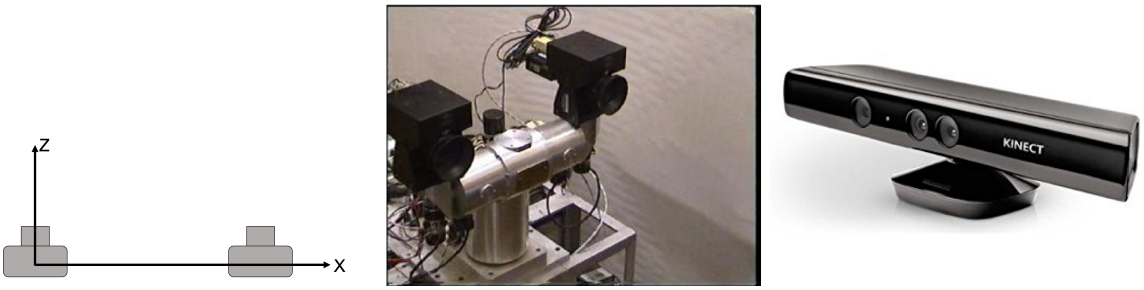

Figure 2: The rectified camera stereo setup (left). An actual stereo camera rig (middle), and the kinect ccamera (right).

This error is the Euclidean distance between the true image location (x_1, y_1), and the predicted image location $(\hat{x}_1(X,Y,Z), \hat{y}_1(X,Y,Z))$ obtained by *reprojecting* the hypothesized world point into the image. As such it is called the *reprojection error*.

One can thus find the 3D world point by minimizing the total squared reprojection error in the two images:

$$\min_{X,Y,Z} (\Delta_1(X,Y,Z)^2 + \Delta_2(X,Y,Z)^2) \tag{14}$$

This is of course a non-linear optimization problem and is thus consequently harder. However, it tends to be (a) more accurate, (b) more interpretable, since the reprojection error is the actual distance between the true and predicted image locations, and (c) more generalizable to other problems in 3D reconstruction.

2 Stereo with rectified cameras

A particular special case of the stereo setup deserves attention. In this special case, the two cameras are oriented parallel to each other, and relative to the first camera, the second camera is translated along the X axis.

Without loss of generality, we can assume that the world coordinate system is aligned with the coordinate system of the first camera and is centered on the first camera pinhole. Thus, in world coordinates, the first camera pinhole is at $(0,0,0)$. If the second camera is b units away along the positive X axis, the second camera pinhole is at $(b,0,0)$. Let us also assume that the matrix K for each camera (i.e., the intrinsic camera parameters) is I. Based on this, we can write down the projection matrices of the two cameras:

$$P^{(1)} \equiv \begin{bmatrix} I & \mathbf{0} \end{bmatrix} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{15}$$

$$P^{(2)} \equiv \begin{bmatrix} I & \begin{bmatrix} -b \\ 0 \\ 0 \end{bmatrix} \end{bmatrix} \equiv \begin{bmatrix} 1 & 0 & 0 & -b \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{16}$$

$$\tag{17}$$

(here we have used the expressions we derived in a previous lecture for the camera projection matrices in terms of the location of its pinhole and the orientation of its axes.)

Given a world point $\vec{\mathbf{Q}} \equiv \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$, we can now figure out where it will project in the two images.

$$\vec{\mathbf{q}}_1 \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \equiv \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{18}$$

$$\vec{\mathbf{q}}_2 \equiv \begin{bmatrix} 1 & 0 & 0 & -b \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \equiv \begin{bmatrix} X - b \\ Y \\ Z \end{bmatrix} \tag{19}$$

Converting both these to Euclidean coordinates (by dividing by the last coordinate and dropping the last coordinate) we get:

$$\mathbf{q}_1 = \begin{bmatrix} \frac{X}{Z} \\ \frac{Y}{Z} \end{bmatrix} \tag{20}$$

$$\mathbf{q}_2 = \begin{bmatrix} \frac{X-b}{Z} \\ \frac{Y}{Z} \end{bmatrix} \tag{21}$$

There are two things to note in this setup:

1. $\mathbf{q}_1$ and $\mathbf{q}_2$ have the same X coordinate. This means that the pair of corresponding pixels must lie on the same row. Thus, to search for corresponding pixels, we do not need to search over the whole image, we can just search along the same row.

2. The difference in X coordinates, d, is given by:

$$d = \frac{b}{Z} \tag{22}$$

   d is called the *disparity*, b is called the *baseline*, Z is the *depth* of the 3D point (how far away from the pinhole it is along the Z axis). Thus:

$$disparity = \frac{baseline}{depth} \tag{23}$$

   Thus, once we find the pair of corresponding pixels, we can get the Z coordinate simply by taking the ratio between the baseline and the disparity. Once we get Z, we can get X and Y easily as well using Equations (20) and (21). Thus estimating the 3D coordinates of a world point from a pair of corresponding pixels is very simple.

The rectified camera setup is therefore simpler both in the estimation of the corresponding pixels as well as in the estimation of the 3D world point. As such, it is very commonly used. If we have control over the location of the cameras, we typically choose this setup. A variant of this setup is also used in multiple *depth sensors* such as the one on the iPhone, or in the older Microsoft Kinecct.

2.1 Estimating disparity

Because of how common the rectified ccamera setup is, algorithms exist specifically for reconstructing 3D points in this setup. These algorithms take as input the images from the two cameras. They then compute a *disparity* value for each pixel in the first image. This is typically represented as a *disparity image*. Once the disparity image is computed, it is trivial to estimate the Z coordinate for each pixel, and then estimate the X and the Y values.

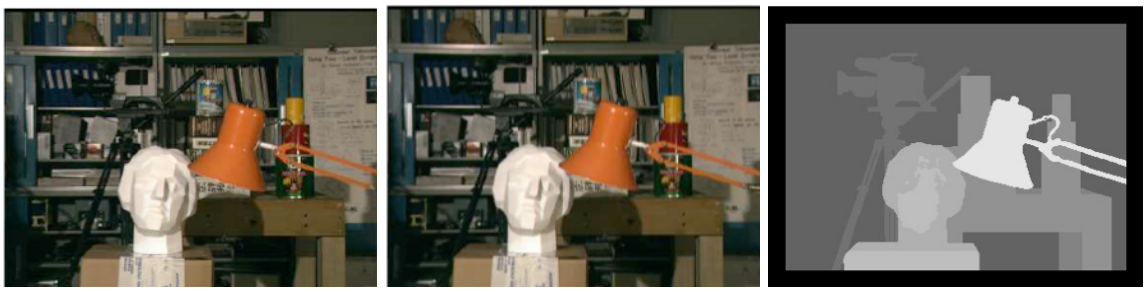

Figure 3: The two images and the disparity image.

To compute the disparity image, we need to iterate over every pixel in image 1, and find the corresponding pixel in image 2. As discussed above, we only need to search in the corresponding row. In addition, because the two images are taken from roughly the same viewpoint and in the same lighting conditions, we often do not need that much invariance to large changes in viewpoint. Therefore, a reasonable choice is to use NCC (*normalized cross correlation*). As discussed before in class, NCC constructs a descriptor for a pixel by taking a patch around it, subtracting off the mean, and then rescaling to unit norm. Two pixels are compared by computing the dot product between these descriptors.

Suppose the two images are $M \times N$. For every pixel (x, y) in image 1, the corresponding pixel must lie on the same row in image 2, somewhere between $(x - D, y)$ and (x, y), where D is the maximum possible disparity (e.g., the width of the image). For each possible disparity value $d \in [0, D]$, we can compute the NCC score between the patch at (x, y) in image 1, and the patch $(x - d, y)$ in image 2. We can store these disparity values in an *NCC cost volume* of size $M \times N \times D$. We can then compute the disparity of each pixel by taking the arg max over the last dimension.

There are two possible ways of computing the NCC cost volume. The first algorithm iterates over pixels, and for each pixel iterates over disparities.

1. For each pixel (x, y) in first image
   (a) For each possible disparity d
      i. Compute NCC between (x, y) in image 1 and $(x - d, y)$ in image 2.

The second algorithm iterates over *disparity values*, and then for each disparity it runs over pixels:

1. For each possible disparity d
   (a) For each pixel (x, y)
      i. Compute NCC between (x, y) in image 1 and $(x - d, y)$ in image 2.

This latter algorithm can end up being simpler to implement, since we are evaluating the same shift d for all pixels together. This in turn can be done by simply shifting the second image by d. Because a disparity value corresponds to a particular depth value, this latter algorithm can be seen as iterating over depths, and then scoring pixels for their likelihood of being at that depth (based on the NCC score). This second algorithm is called *plane-sweep stereo*.

Both algorithms are equivalent, in that they produce the same NCC cost volume. Taking the arg max over the last dimension then gives us the disparities and thus the depths.