DRAFT: Computer Vision for the Uninitiated

Sudeep Sarkar

February 24, 2020

Chapter 1

Segmentation using Mean Shift

- Section 5.3.2 of text book and paper by Peter Meer
- Core algorithmic task: Efficiently find the peaks of a distribution given a set of samples.
- Let $\mathbf{x}_i$ represent the vector representing a pixel in terms of color or other attributes.
- Kernel distribution is an estimate of a continuous probability density function from samples. It is achieved by replacing a continuous function, which we will call the kernel function $G(\mathbf{x})$ at sample and adding them up. It effectively gives us a continuous interpolation of the data between the samples. The most used kernel function is a Gaussian distribution.

$$f(\mathbf{x}) = c\sum_{i=1}^{N} G(\mathbf{x} - \mathbf{x}_i) = c\sum_{i=1}^{N} \exp\left(-\frac{||\mathbf{x} - \mathbf{x}_i||^2}{2h^2}\right) \tag{1.1}$$

  The constant c is a normalizing constant to make the function integrate out to one. For a vector valued sample, this is basically a sum of Gaussian balls centered on the samples.

- **Multiple restart gradient ascent:** Our approach to finding the peaks in this continuous function would be to start from a random point and then to move it iteratively towards a peak. Ideally, we would like to start from many different points so that we are able to that converge around all the available peaks. How many samples

should we start from? That is an issue that is needs to be looked at during implementation time. This is a kin to hill climbing but from many different points.

Another way to visualize this process is to imagine throwing a random number of particles on this continuous function and then moving these particles up the hill wherever they are placed. The idea is that overtime the particles will accumulate on the hilltops nearest to them. Let us first see how we can quantify the step we need to take at each iteration from each particle or starting point.

- For each particle (or estimate of a possible peak, $\mathbf{y}_k$, we modify it by an uphill step, which is estimated by the gradient of the function at $\mathbf{y}_k$.
- So, first we need to an expression of the gradient of the continuous function $f(\mathbf{x})$. Note that the function is a scalar value define over a vector field.
- Worksheet problem: Draw the Gaussian function. Draw the first derivative of a Gaussian. Express the gradient of a one dimensional Gaussian kernel function and plot the function. Let

$$f(x) = c\sum_{i=1}^{N} \exp\left(-\frac{(x-x_i)^2}{2h^2}\right) \tag{1.2}$$

$$f^{'}(x) = c\frac{df(x)}{dx} = \sum_{i=1}^{N} -\frac{x-x_i}{h^2}\exp\left(\frac{(x-x_i)^2}{2h^2}\right) \tag{1.3}$$

Note that this is equivalent to placing a first derivative of a Gaussian function at each sample point.

-

$$\nabla f(\mathbf{x}) = c\sum_{i=1}^{N} -\frac{\mathbf{x}-\mathbf{x}_i}{h^2}\exp\left(\frac{||\mathbf{x}-\mathbf{x}_i||^2}{2h^2}\right) \tag{1.4}$$

$$= c\sum_{i=1}^{N} -\frac{\mathbf{x}-\mathbf{x}_i}{h^2}G(\mathbf{x}-\mathbf{x}_i) \tag{1.5}$$

•

$$\begin{aligned}
\nabla f(\mathbf{x}) &= \frac{c}{h^2}\sum_{i=1}^{N} -(\mathbf{x}-\mathbf{x}_i)G(\mathbf{x}-\mathbf{x}_i) && (1.6)\\
&= \frac{c}{h^2}\sum_{i=1}^{N} -\mathbf{x}G(\mathbf{x}-\mathbf{x}_i)+\mathbf{x}_iG(\mathbf{x}-\mathbf{x}_i) && (1.7)\\
&= \frac{c}{h^2}\left(-\mathbf{x}\sum_{i=1}^{N} G(\mathbf{x}-\mathbf{x}_i)+\sum_{i=1}^{N}\mathbf{x}_iG(\mathbf{x}-\mathbf{x}_i)\right) && (1.8)\\
&= \frac{c}{h^2}\left(\sum_{i=1}^{N} G(\mathbf{x}-\mathbf{x}_i)\right)\left(-\mathbf{x}+\frac{\sum_{i=1}^{N}\mathbf{x}_iG(\mathbf{x}-\mathbf{x}_i)}{\sum_{i=1}^{N}G(\mathbf{x}-\mathbf{x}_i)}\right) && (1.9)\\
&= \left[\frac{c}{h^2}\sum_{i=1}^{N} G(\mathbf{x}-\mathbf{x}_i)\right]\left(\frac{\sum_{i=1}^{N}\mathbf{x}_iG(\mathbf{x}-\mathbf{x}_i)}{\sum_{i=1}^{N}G(\mathbf{x}-\mathbf{x}_i)}-\mathbf{x}\right) && (1.10)
\end{aligned}$$

- Note that in the last expression above, the term in the square bracket is a scalar quantity, while the term in parenthesis is a vector. The direction of the vector is the direction of the gradient. We will denote this vector by $\mathbf{m}(\mathbf{x})$.

$$\mathbf{m}(\mathbf{x}) = \left(\frac{\sum_{i=1}^{N}\mathbf{x}_iG(\mathbf{x}-\mathbf{x}_i)}{\sum_{i=1}^{N}G(\mathbf{x}-\mathbf{x}_i)}-\mathbf{x}\right) \tag{1.11}$$

- Now that we have an expression for the gradient vector, we can use it to modify the current estimate of the peak denoted by $\mathbf{y}_k$.

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \mathbf{m}(\mathbf{y}_k) = \frac{\sum_{i=1}^{N}\mathbf{x}_iG(\mathbf{y}_k-\mathbf{x}_i)}{\sum_{i=1}^{N}G(\mathbf{y}_k-\mathbf{x}_i)} \tag{1.12}$$

- The above question is basically saying that the new estimate of the peak location is the weighted mean of the sample, $\mathbf{x}_i$. The weights are local because the Gaussian function is locally weighted or has high values only with a small region around the center.

- A possible in class assignment would be to draw the Gaussian functions over each sample and see how the above expression is really a weighted mean of the local values.

- Intuitively, the above method works because by definition you have more samples around the (histogram) peaks and a random starting point will be pulled towards denser set of points.

- Although the above is a derivation for the Gaussian kernel, Comaniciu and Meer (2002) has shown that this also works for other kernel functions as long as they are monotonically decreasing with distance from the center. Another kernel functions studied by them was the Epamechnikov kernel that is defined over the unit ball given by:

$$E(\mathbf{x} - \mathbf{x}_i) = \max\left(0, 1 - ||\mathbf{x} - \mathbf{x}_i||^2\right) \tag{1.13}$$

- Possible worksheet: What is the $\mathbf{m}(\mathbf{x})$, the direction of the gradient at any point, for this kernel? What is the update equation?

- There are many methods that have been proposed in the literature to make this the update process more efficient and fast. The speed depends on the number of pixels, which can be very large for big images. Complexity is $\mathcal{O}(N^2)$, where N is the number of pixels.

- To prevent clustering of accidentally similar colored pixels from distant parts of the image, in practice, one can expand the pixel value vector to include the location (x, y). We have to use different values for the radius h (kernel width) for the different dimensions as they represent different things. We use h_s for the spatial dimension and h_r for the color dimension.

$$\begin{aligned} G(\mathbf{x} - \mathbf{x}_i) &= \exp\left(-\frac{||\mathbf{x} - \mathbf{x}_i||^2}{2h^2}\right) && (1.14) \\ &= \exp\left(\frac{(x - x_i)^2 + (y - y_i)^2}{2h_s^2}\right) && (1.15) \\ &\quad + \exp\left(\frac{(r - r_i)^2 + (g - g_i)^2 + +(b - b_i)^2}{2h_r^2}\right) && (1.16) \end{aligned}$$

- The choice of the parameters h_s and h_r is by trial and error. Literature survey will reveal several strategies.

Chapter 2

Linear Filtering and Edges

- Section 3.2, correlation, convolution, Gaussian smoothing, 2D filtering using 1D filters.
- Linear filtering is a basic operation done on the image for a variety of purposes, to reduce noise, sharpen features, emphasize certain kinds of features such as edges, corners, etc. See Fig 3.11 for examples.
- Most common version is the weighted sum of neighboring pixels. This is called a **correlation** given by

$$\begin{aligned} g(i,j) &= h \otimes f && (2.1)\\ &= \sum_{k=1}^{N}\sum_{l=1}^{N} h(k,l) f(i+k, j+l) && (2.2)\\ &= f \otimes h && (2.3)\\ &= \sum_{k,l} f(k,l) h(i+k, j+l) && (2.4) \end{aligned}$$

  The function $h(i,j)$ is called a filter or mask. Place the filter at a location, multiple, and then add to get the value. Note that the operation is commutative. So, equivalently shift the image, keeping the filter fixed, and then multiply and add.

- The **convolution** flips the filter about the origin and then correlates with the image function.

$$f \star h = \sum_{k,l} f(k,l)h(i-k, j-l) \tag{2.5}$$

$$h \star f = \sum_{k,l} h(k,l)f(i-k, j-l) \tag{2.6}$$

For circularly symmetric filters, convolution is the same as correlation.

- **Worksheet:** Let $h(i,j)$ be a zero valued mask with only one non-zero entry at the center equal to 1. What is the convolution of any image with this mask?

- Convolution is a linear operation: convolution of the sum of two images is the sum of the convolution of each of them. Convolution of a scaled version of a image is the scaled version of the convolution of the image.

$$h \star (af_1 + f_2) = a(h \star f_1) + (h \star f_2) \tag{2.7}$$

- Convolution is a shift invariant operation since the filter is the same for each shift.

- So, convolution is a linear, shift-invariant (LSI) operation.

- Continuous version

$$f \star h = \int_u \int_v f(u,v)h(x-u, y-v)dudv \tag{2.8}$$

$$= \int_u \int_v h(u,v)f(x-u, y-v)dudv \tag{2.9}$$

- Derivative of a convolution is the convolution.

$$\frac{\partial}{\partial x} f \star h = \int_u \int_v f(u,v) \left(\frac{\partial}{\partial x} h(x-u, y-v) \right) dudv \tag{2.10}$$

$$= \int_u \int_v f(u,v)h_x(x-u, y-v)dudv \tag{2.11}$$

- Filtering with Gaussian shaped filters. Consider just one dimensional filtering for now. Let the filter be

$$h(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp -\frac{x^2}{2\sigma^2} \tag{2.12}$$

And let the "image" be a step function $u(x)$.

$$s(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{2.13}$$

- $g(x) = s(x) \star h(x)$ will smooth the sharp edge.

$$\begin{aligned} s(x) \star h(x) &= \int_u s(u)h(x-u)du & (2.14) \\ &= \int_{u=0}^{\infty} h(x-u)du & (2.15) \\ &= \int_{v=-\infty}^{x} h(v)dv & (2.16) \end{aligned}$$

- The derivative of the output is a Gaussian function. We can use this operation, i.e. convolution with a Gaussian followed by a derivative for "edge" detection. The point of maximum marks the edge. This is equivalent to convolution with a derivative of a Gaussian function.

$$h'(x) = -\frac{x}{\sqrt{2\pi}\sigma^3} \exp -\frac{x^2}{2\sigma^2} \tag{2.17}$$

- Taken another derivative produces a zero at the edge location. This is equivalent to convolution with a second derivative of a Gaussian.

$$\begin{aligned} h''(x) &= -\frac{1}{\sqrt{2\pi}\sigma^3} \exp -\frac{x^2}{2\sigma^2} + \frac{x^2}{\sqrt{2\pi}\sigma^5} \exp -\frac{x^2}{2\sigma^2} & (2.18) \\ &= -\frac{1}{\sqrt{2\pi}\sigma^3} \left(1 - \frac{x^2}{\sigma^2}\right) \exp -\frac{x^2}{2\sigma^2} & (2.19) \end{aligned}$$

- **Worksheet:** Plot the above function.

- The 2D equivalent is the Laplacian of a Gaussian

$$\nabla h(x,y) = \frac{\partial^2}{\partial x^2} h(x,y) + \frac{\partial^2}{\partial y^2} h(x,y) \tag{2.20}$$

- 2D convolution by Gaussian is separable.

$$h(x,y) = \frac{1}{2\pi\sigma^2}\exp -\frac{x^2+y^2}{2\sigma^2} \quad (2.21)$$
$$= h(x)h(y) \quad (2.22)$$

$$f(x,y) \star h(x,y) = (f(x,y) \star h(x)) \star h(y) \quad (2.23)$$

Chapter 3

Point Features

- Figure 4.2 and 4.3
- Point features are stable with respect to viewpoint and lighting.
- To track points across video frames possible matching criterion is weighted image difference.

$$E(\mathbf{u}) = \sum_i w(\mathbf{x}_i) \left(I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)\right)^2 \tag{3.1}$$

- The value of this would be low when regions match, but could be low for other reason too, such as dark patches.
- How can one select points to track? Find points in image such as the error criterion is stable with respect to small variations, on the same image. (see Fig 4.5)

$$E_{AC}(\triangle\mathbf{u}) = \sum_i w(\mathbf{x}_i) \left(I_0(\mathbf{x}_i + \triangle\mathbf{u}) - I_0(\mathbf{x}_i)\right)^2 \tag{3.2}$$

- Using Taylor series expansion

$$\begin{aligned} I_0(\mathbf{x}_i + \triangle\mathbf{u}) &\approx I_0(\mathbf{x}_i) + \bigtriangledown I_0(\mathbf{x}_i)^T \Delta\mathbf{u} && (3.3) \\ &= I_0(\mathbf{x}_i) + \begin{bmatrix} I_x(\mathbf{x}_i) & I_y(\mathbf{x}_i) \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle y \end{bmatrix} && (3.4) \end{aligned}$$

- The error criterion on the same image can be approximated using the Taylor expansion above.

$$
\begin{aligned}
E_{AC}(\triangle \mathbf{u}) &= \sum_i w(\mathbf{x}_i) \left(I_0(\mathbf{x}_i + \triangle \mathbf{u}) - I_0(\mathbf{x}_i)\right)^2 && (3.5)\\
&\approx \sum_i w(\mathbf{x}_i) \left(\bigtriangledown I_0(\mathbf{x}_i)^T \Delta \mathbf{u}\right)^2 && (3.6)\\
&= \sum_i w(\mathbf{x}_i) \left(\bigtriangledown I_0(\mathbf{x}_i)^T \Delta \mathbf{u}\right)^T \left(\bigtriangledown I_0(\mathbf{x}_i)^T \Delta \mathbf{u}\right) && (3.7)\\
&= \sum_i w(\mathbf{x}_i) \begin{bmatrix}\triangle x & \triangle y\end{bmatrix} \begin{bmatrix} I_x(\mathbf{x}_i) \\ I_y(\mathbf{x}_i)\end{bmatrix} \begin{bmatrix} I_x(\mathbf{x}_i) & I_y(\mathbf{x}_i)\end{bmatrix} \begin{bmatrix}\triangle x \\ \triangle y\end{bmatrix} && (3.8)\\
&= \begin{bmatrix}\triangle x & \triangle y\end{bmatrix} \left(\sum_i w(\mathbf{x}_i) \begin{bmatrix} I_x(\mathbf{x}_i) \\ I_y(\mathbf{x}_i)\end{bmatrix} \begin{bmatrix} I_x(\mathbf{x}_i) & I_y(\mathbf{x}_i)\end{bmatrix} \right) \begin{bmatrix}\triangle x \\ \triangle y\end{bmatrix} && (3.9)\\
&= \begin{bmatrix}\triangle x & \triangle y\end{bmatrix} \mathbf{A} \begin{bmatrix}\triangle x \\ \triangle y\end{bmatrix} && (3.10)
\end{aligned}
$$

$$
\mathbf{A} = \begin{bmatrix} \sum_i w(\mathbf{x}_i) I_x^2(\mathbf{x}_i) & \sum_i w(\mathbf{x}_i) I_x(\mathbf{x}_i) I_y(\mathbf{x}_i) \\ \sum_i w(\mathbf{x}_i) I_y(\mathbf{x}_i) I_x(\mathbf{x}_i) & \sum_i w(\mathbf{x}_i) I_y^2(\mathbf{x}_i) \end{bmatrix} \tag{3.11}
$$

$$
\mathbf{A} = \begin{bmatrix} \sum_i w(\mathbf{x}_i) I_x^2 & \sum_i w(\mathbf{x}_i) I_x I_y \\ \sum_i w(\mathbf{x}_i) I_y I_x & \sum_i w(\mathbf{x}_i) I_y^2 \end{bmatrix} \tag{3.12}
$$

$$
\mathbf{A} = \sum_i w(\mathbf{x}_i) \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix} \tag{3.13}
$$

- The matrix $\mathbf{A}$ is an estimate of the local image tensor.
- $\mathbf{A}^{-1}$ is a measure of the uncertainty of the location, i.e. how sharp the valley is?
- If λ_0 and λ_1 are eigenvalues of A, where $\lambda_0 < \lambda_1$, then λ_0^{-1} and λ_1^{-1} are the eigenvalues of $\mathbf{A}^{-1}$. The largest eigenvalue of $\mathbf{A}^{-1}$ should be small, i.e. λ_0^{-1} should be small, or in other words λ_0 should be high (Shi and Tomasi).
- $\lambda_0 \lambda_1 = \det(\mathbf{A})$ and $\lambda_0 + \lambda_1 = Trace(\mathbf{A})$

- For Harris points: $\lambda_0\lambda_1 - \alpha(\lambda_0+\lambda_1)^2$ should be high, typically $\alpha = 0.06$
- Others suggest the harmonic mean, $\frac{\lambda_0\lambda_1}{\lambda_0+\lambda_1}$
- Feature tracking - Section 4.1.4. We expect motion from frame to frame is small.
- Detect stable features in one frame.
- Correlate patches over small neighborhoods and pick point of smallest squared difference.
- If features are tracked over long sequences, appearance can change, so need to update template, but not too often.

Chapter 4

Kalman Tracking

- Consider the task of tracking moving stuff in videos taken from fixed cameras. This is the simplest imaging model we can consider for this problem. Whatever is developed here can be generalized and formulated for other complex cases such as moving cameras with moving objects. Of course, the equations in the world will be more complicated.
- We are going to differentiate between the measurement we make about the moving objects and the state of the moving objects. The states of the moving objects are not directly observed. We can make measurements about some of the components of the states of the moving objects.
- Let $\mathbf{s}_t$ denote the state of the moving objects at time t. Possible choices of the state vector are: just location, $\begin{bmatrix} x & y \end{bmatrix}^T$ or location and velocity $\begin{bmatrix} x & y & v_x & v_y \end{bmatrix}^T$ or location, velocity, and bounding box size $\begin{bmatrix} x & y & w & h & v_x & v_y \end{bmatrix}^T$ or location, velocity, acceleration, and bounding box size $\begin{bmatrix} x & y & w & h & v_x & v_y & a_x & a_y \end{bmatrix}^T$
- This state is really a random variable as we do not know the exact value. We assume it is a Gaussian random variable, with a mean value, $\hat{\mathbf{s}}_t$, and a covariance $\mathbf{P}_t$ capturing the uncertainty about the state.
- A measurement model relates the measurement with the underlying model states. In general the dimension of the measurement vector

would be lower than the state vector. For instance, we can measure the location directly, but not the velocity or acceleration; so the measurement vector in that case will consist of just the location values.

Let the measurement vector be $\mathbf{m}_t$ and the relationship with the state vector $\mathbf{s}_t$ be related by

$$\mathbf{m}_t = \mathbf{H}\mathbf{s}_t + \mathbf{w} \tag{4.1}$$

where $\mathbf{w}$ is the measurement error, which again will be assume to zero mean with some covariance, $\mathbf{R}$, capturing the quality of the measuring modality. For instance, if you are measuring the location sum of squared differences between patches from earlier point features, you can use the inverse of the local image tensor (Eq. 3.13) as a measure of uncertainty.

One example of a measurement model is given by.

$$\begin{bmatrix} r \\ c \end{bmatrix}_{t+1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}_t + \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \tag{4.2}$$

- We assume a linear model of the state change.

$$\mathbf{s}_{t+1} = \mathbf{A}\mathbf{s}_t + \mathbf{n} \tag{4.3}$$

where $\mathbf{n}$ is zero mean noise with covariance $\mathbf{Q}$. Note that $\mathbf{s}_{t+1}$ is linear combination of two Gaussian random variable so it is also a Gaussian random variable. One example of a constant velocity model is given by

$$\begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}_{t+1} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}_t + \begin{bmatrix} n_0 \\ n_1 \\ n_2 \\ n_3 \end{bmatrix} \tag{4.4}$$

- At each time instant, t, the following variables are updated: the mean value of the state $\hat{\mathbf{s}}_t$ and its covariance $\mathbf{P}$. These quantities are effected by the state evolution function and also indirectly by the measurement. So, the update is in two steps.

- **Forward Update:** Update based on the state equation. Note the noise is zero measure and remember the addition of Gaussian random variables is also a Gaussian. Let $\mathbf{Y} = \mathbf{X}_1 + \mathbf{X}_2$, where each of them is Gaussian random variable. Then $\widehat{\mathbf{Y}} = \widehat{\mathbf{X}}_\mathbf{1} + \widehat{\mathbf{X}}_\mathbf{2}$ and $\mathbf{\Sigma}_Y = \mathbf{\Sigma}_{X_1} + \mathbf{\Sigma}_{X_2}$.

  The intermediate estimate of the new state and its covariance, just based on the forward model is given by

$$\hat{\mathbf{s}}_{t+1}^{-} = \mathbf{A}\hat{\mathbf{s}}_t \tag{4.5}$$
$$\mathbf{P}_{t+1}^{-} = \mathbf{A}\mathbf{P}_t\mathbf{A}^T + \mathbf{Q} \tag{4.6}$$

- **Kalman Gain:** blends the model covariance and measurement error covariance

$$\mathbf{K} = \mathbf{P}_{t+1}^{-}\mathbf{H}^T\left(\mathbf{H}\mathbf{P}_{t+1}^{-}\mathbf{H}^T + \mathbf{R}\right)^{-1} \tag{4.7}$$

- **Measurement Update:**

$$\hat{\mathbf{s}}_{t+1} = \hat{\mathbf{s}}_{t+1}^{-} + \mathbf{K}\left(\mathbf{m}_{t+1} - \mathbf{H}\hat{\mathbf{s}}_{t+1}^{-}\right) \tag{4.8}$$
$$\mathbf{P}_{t+1} = \left(\mathbf{I} - \mathbf{K}\mathbf{H}\right)\mathbf{P}_{t+1}^{-} \tag{4.9}$$

  Note how the measurement error, i.e. the terms in parenthesis, is added to the state to update it.

- Work through the following example with constant velocity model example shown earlier, with the following values of the other variables you would need.

  We assume half a pixel model error noise, which implies that we have faith in the model.

$$\mathbf{Q} = \begin{bmatrix} \frac{1}{4} & & 0 & \\ & \frac{1}{4} & & \\ & & \frac{1}{4} & \\ 0 & & & \frac{1}{4} \end{bmatrix} \tag{4.10}$$

  Initial estimate of the covariance model assumes that it is diagonal, uncorrelated dimensions, with 3 pixel standard deviation in location and 5 pixel standard deviation in velocity.

$$\mathbf{P}_0 = \begin{bmatrix} 9 & & 0 & \\ & 9 & & \\ & & 25 & \\ 0 & & & 25 \end{bmatrix} \tag{4.11}$$

The measurement error is diagonal too with a standard deviation of 1 pixel.

$$\mathbf{R} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.12}$$

Let $\hat{\mathbf{s}}_0 = \begin{bmatrix} 100 \\ 170 \\ 0 \\ 0 \end{bmatrix}$ and measurement be $\mathbf{m}_1 = \begin{bmatrix} 103 \\ 163 \end{bmatrix}$, what is $\mathbf{s}_1$?

$$\hat{\mathbf{s}}_1^- = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 100 \\ 170 \\ 0 \\ 0 \end{bmatrix} \tag{4.13}$$

$$= \begin{bmatrix} 100 \\ 170 \\ 0 \\ 0 \end{bmatrix} \tag{4.14}$$

$$\mathbf{P}_1^- = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 9 & & 0 & \\ & 9 & & \\ & & 25 & \\ 0 & & & 25 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} + \begin{bmatrix} \frac{1}{4} & & 0 & \\ & \frac{1}{4} & & \\ & & \frac{1}{4} & \\ 0 & & & \frac{1}{4} \end{bmatrix} \tag{4.15}$$

$$= \begin{bmatrix} 34.25 & 0 & 25 & 0 \\ 0 & 34.25 & 0 & 25 \\ 25 & 0 & 25.25 & 0 \\ 0 & 2 & 0 & 25.25 \end{bmatrix} \tag{4.16}$$

This state prediction gives us a location to relocate the new location and the covariance estimate gives us an area to consider. In our case this is a circle centered at $(100, 170)$ and with radius $\sqrt{34.25} = 5.85$ pixels. We can run the SSD detector and make a new measurement. $\mathbf{m}_1 = \begin{bmatrix} 103 \\ 163 \end{bmatrix}$

$$\mathbf{K} = \begin{bmatrix} 0.972 & 0 \\ 0 & 0.972 \\ 0.709 & 0 \\ 0 & 0.709 \end{bmatrix} \tag{4.17}$$

The entries capture how much to weight the measurement. Values close to 1 suggest trust the measurement.

$$\hat{\mathbf{s}}_1 = \hat{\mathbf{s}}_1^- + \mathbf{K}\left(\mathbf{m}_1 - \mathbf{H}\hat{\mathbf{s}}_1^-\right) \tag{4.18}$$

$$= \begin{bmatrix} 100 \\ 170 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.972 & 0 \\ 0 & 0.972 \\ 0.709 & 0 \\ 0 & 0.709 \end{bmatrix} \left(\begin{bmatrix} 103 \\ 163 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 100 \\ 170 \\ 0 \\ 0 \end{bmatrix} \right) \tag{4.19}$$

$$= \begin{bmatrix} 102.9 \\ 163.2 \\ 2.13 \\ -4.96 \end{bmatrix} \tag{4.20}$$

$$\mathbf{P}_1 = (\mathbf{I} - \mathbf{K}\mathbf{H})\,\mathbf{P}_1^- \tag{4.21}$$

$$= \left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0.972 & 0 \\ 0 & 0.972 \\ 0.709 & 0 \\ 0 & 0.709 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \right) \begin{bmatrix} 34.25 & 0 & 25 & 0 \\ 0 & 34.25 & 0 & 25 \\ 25 & 0 & 25.25 & 0 \\ 0 & 2 & 0 & 25.25 \end{bmatrix} \tag{4.22}$$

$$= \begin{bmatrix} 0.971 & 0 & 0.71 & 0 \\ 0 & 0.971 & 0 & 0.71 \\ 0.71 & 0 & 7.52 & 0 \\ 0 & 0.71 & 0 & 7.52 \end{bmatrix} \tag{4.23}$$

Note how the covariance matrix is not diagonal anymore.

Chapter 5

2D Geometric Transformation

See Fig 4.2 – observe the geometric transformation between the planar surfaces between the two views. What is the mathematical relationship between them? How can we match image regions in a geometrically consistent manner? If you are tracking points on a plane, what model can we use as the underlying state model? What are some of the geometric models we can try? We will start with 2D transformations.

- **Homogeneous coordinates of points:** A 2D points, $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$, is mapped to a 3D line through the origin given by $\widetilde{\mathbf{x}} = w \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$. We can see that we can transform a 2D representation to a 3D representation and vice-versa. We will see that expressing the points in this way makes the representation of the transformations simpler, notation wise. Many transformations can be expressed as matrix multiplication, which makes it easier to represent a chain of transformation as simple matrix multiplication.
- **Homogeneous coordinates of lines:** Equation of a 2D line can be written as $ax + by + c = 0$ or as $\begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$ or as $\widetilde{\mathbf{n}}^T \widetilde{\mathbf{x}} = 0$, where $\widetilde{\mathbf{n}}$ is the homogeneous representation of a line. Note that the homogeneous representation of a line is the same as a point. Scaled

version of the homogeneous representation, $w\widetilde{\mathbf{n}}$ represents the same line, as $wax + wby + wc = w(ax + by + c) = 0$

- Figure 2.4 and Table 2.1
- **Translation:** preserves orientation, angles, and length

$$\mathbf{x}' = \mathbf{x} + \mathbf{t} \tag{5.1}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{5.2}$$

We can express the same in homogeneous coordinates as:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.3}$$

$$\widetilde{\mathbf{x}}' = \begin{bmatrix} \mathbf{I}^{2\times 2} & \mathbf{t}^{2\times 1} \\ \mathbf{0}^{1\times 2} & 1 \end{bmatrix} \widetilde{\mathbf{x}} \tag{5.4}$$

- **Rotation and translation:** preserves angles and length

$$\mathbf{x}' = \mathbf{R}^{2\times 2}\mathbf{x} + \mathbf{t} \tag{5.5}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{5.6}$$

We can express the same in homogeneous coordinates as:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & t_x \\ \sin(\theta) & \cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.7}$$

$$\widetilde{\mathbf{x}}' = \begin{bmatrix} \mathbf{R}^{2\times 2} & \mathbf{t}^{2\times 1} \\ \mathbf{0}^{1\times 2} & 1 \end{bmatrix} \widetilde{\mathbf{x}} \tag{5.8}$$

The rotation matrix is not any 2×2 matrix. It has certain properties it has to satisfy: $\mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I}$ and $\det(\mathbf{R}) = 1$. The columns and rows are orthogonal to each other and their individual magnitude is one. In other words, the matrix is an **orthonormal** matrix.

- **Similarity Transformation - scaled rotation and translation:** preserves angles only

$$\mathbf{x}' = s\mathbf{R}\mathbf{x} + \mathbf{t} \tag{5.9}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s\cos(\theta) & -s\sin(\theta) \\ s\sin(\theta) & s\cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{5.10}$$

We can express the same in homogeneous coordinates as:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s\cos(\theta) & -s\sin(\theta) & t_x \\ s\sin(\theta) & s\cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.11}$$

$$\widetilde{\mathbf{x}}' = \begin{bmatrix} \begin{bmatrix} a & -b \\ b & a \end{bmatrix} & \mathbf{t}^{2\times 1} \\ \mathbf{0}^{1\times 2} & 1 \end{bmatrix} \widetilde{\mathbf{x}} \tag{5.12}$$

Note that unlike for a rotation matrix: $a^2 + b^2 \neq 1$.

- **Affine:** preserves only parallelism and changes orientation, angles, and lengths

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & t_x \\ a_{10} & a_{11} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.13}$$

Note that the "rotation" matrix is allowed to be any possible 2×2 matrix.

- WS: How would you show that affine transformation preserves parallel lines?

- **Homography:** preserves only straight lines, i,e. straight lines gets matched to straight lines, but other properties are not preserved including parallelism. This is the geometric model relating two perspective views of a plane. We will study this later.

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.14}$$

$$\widetilde{\mathbf{x}'} = \widetilde{\mathbf{H}}\widetilde{\mathbf{x}} \tag{5.15}$$

$$\mathbf{x} = \begin{bmatrix} a/c \\ b/c \end{bmatrix} \tag{5.16}$$

- Lines are also transformed into lines by homography. Similar relationship can be derived for the other transformations.

$$\widetilde{\mathbf{x}'} = \widetilde{\mathbf{H}}\widetilde{\mathbf{x}} \tag{5.17}$$

$$\widetilde{\mathbf{x}} = \widetilde{\mathbf{H}}^{-1}\widetilde{\mathbf{x}'} \tag{5.18}$$

$$\widetilde{\mathbf{n}}^T\widetilde{\mathbf{x}} = 0 \tag{5.19}$$

$$\widetilde{\mathbf{n}}^T\widetilde{\mathbf{H}}^{-1}\widetilde{\mathbf{x}'} = 0 \tag{5.20}$$

$$(\widetilde{\mathbf{n}}^T\widetilde{\mathbf{H}}^{-1})\widetilde{\mathbf{x}'} = 0 \tag{5.21}$$

$$(\widetilde{\mathbf{H}}^{-T}\widetilde{\mathbf{n}})^T\widetilde{\mathbf{x}'} = 0 \tag{5.22}$$

$$\widetilde{\mathbf{n}'} = (\widetilde{\mathbf{H}}^{-T}\widetilde{\mathbf{n}}) \tag{5.23}$$

- **Stretch** and **squash** transformations are restricted forms of affine transformation.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00} & 0 & t_x \\ 0 & a_{11} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.24}$$

- These transformations (with some restrictions on degenerate values) form closed groups. In fact, they form a nested set of groups. A group is a set, points in 2D in our case, together with an operation, the transformation, and exhibit the four properties: closure, associativity, identity and invertibility. Which ones exhibit this property?

  Rigid transformations form a group as it is invertible. Not all affine and homography are invertible. If we restrict these transformations to be invertible then they form groups. They all have the other three properties.

We will use the following notation to generically represent these transformations.

$$f(\widetilde{\mathbf{x}}_\mathbf{i}; \mathbf{p}) \tag{5.25}$$

where $\mathbf{p}$ is the set of parameters to be estimated.

For numerical convenience of choosing initialization, we usually reparameterize the transformations so that when the parameters, $\mathbf{p}$, are all zero, the point does not move. To achieve this we add a 1 to the first two diagonal values of the transformation matrix. See Table 6.1

5.1 2D affine alignment

Given a set of corresponding points between two 2D patches, $\{(\widetilde{\mathbf{x}}_\mathbf{i}', \widetilde{\mathbf{x}}_\mathbf{i}) | i = 1, \cdots n\}$ find the transformation between them. We will first consider the affine transformation, which is linear with respect to the parameters. Note this concept of linearity is different from the transformation being linear.

We have three points to consider: the initial point $\widetilde{\mathbf{x}}_\mathbf{i}$, the transformed point using the transformation, $f(\widetilde{\mathbf{x}}_\mathbf{i}; \mathbf{p})$, and the final point, $\widetilde{\mathbf{x}}_\mathbf{i}'$.

The residual (the error) is given by

$$\begin{aligned}
\mathbf{r}_i &= f(\widetilde{\mathbf{x}}_\mathbf{i}; \mathbf{p}) - \widetilde{\mathbf{x}}_\mathbf{i}' && (5.26)\\
&= \begin{bmatrix} a_{00}+1 & a_{01} & t_x \\ a_{10} & a_{11}+1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} - \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} && (5.27)\\
&= \left(\begin{bmatrix} a_{00} & a_{01} & t_x \\ a_{10} & a_{11} & t_y \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} - \begin{bmatrix} \hat{x}_i' \\ y_i' \\ 1 \end{bmatrix} && (5.28)\\
&= \begin{bmatrix} a_{00} & a_{01} & t_x \\ a_{10} & a_{11} & t_y \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} - \begin{bmatrix} x_i' - x_i \\ y_i' - y_i \\ 0 \end{bmatrix} && (5.29)\\
&= \begin{bmatrix} a_{00} & a_{01} & t_x \\ a_{10} & a_{11} & t_y \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} - \begin{bmatrix} x_i' - x_i \\ y_i' - y_i \end{bmatrix} && (5.30)\\
\mathbf{r}_i^{2\times 1} &= \begin{bmatrix} 1 & 0 & x_i & y_i & 0 & 0 \\ 0 & 1 & 0 & 0 & x_i & y_i \end{bmatrix} \begin{bmatrix} t_x \\ t_y \\ a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{bmatrix} - \begin{bmatrix} x_i' - x_i \\ y_i' - y_i \end{bmatrix} && (5.31)\\
&= J(\mathbf{x_i})\mathbf{p} - \Delta\mathbf{x_i} && (5.32)
\end{aligned}$$

Note that $\Delta\mathbf{x_i} = (\mathbf{x_i}' - \mathbf{x_i})$ is the difference between initial and new point.

The term $J(\mathbf{x_i})$ is called the Jacobian and you can see is the derivative of the function $f(\widetilde{\mathbf{x}}_\mathbf{i}; \mathbf{p})$ with respect to $\mathbf{p}$.

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{p}} f(\widetilde{\mathbf{x}}_\mathbf{i}; \mathbf{p}) &= \begin{bmatrix} \frac{\partial f_x}{\partial p_0} & \frac{\partial f_x}{\partial p_1} & \cdots & \frac{\partial f_x}{\partial p_n} \\ \frac{\partial f_y}{\partial p_0} & \frac{\partial f_x}{\partial p_1} & \cdots & \frac{\partial f_y}{\partial p_n} \end{bmatrix} && (5.33)\\
&= \begin{bmatrix} 1 & 0 & x_i & y_i & 0 & 0 \\ 0 & 1 & 0 & 0 & x_i & y_i \end{bmatrix} && (5.34)
\end{aligned}$$

The total residual is given by

$$
\begin{aligned}
r &= \sum_i \|\mathbf{r}_i\|^2 && (5.35)\\
&= \sum_i \mathbf{r}_i^T \mathbf{r}_i = \sum_i \begin{bmatrix} - & - \end{bmatrix} \begin{bmatrix} - \\ - \end{bmatrix} && (5.36)\\
&= \sum_i \left(J(\mathbf{x_i})\mathbf{p} - \Delta\mathbf{x_i}\right)^T \left(J(\mathbf{x_i})\mathbf{p} - \Delta\mathbf{x_i}\right) && (5.37)\\
&= \sum_i \left(\mathbf{p}^T J(\mathbf{x_i})^T - \Delta\mathbf{x_i}^T\right) \left(J(\mathbf{x_i})\mathbf{p} - \Delta\mathbf{x_i}\right) && (5.38)\\
&= \sum_i \mathbf{p}^T J(\mathbf{x_i})^T J(\mathbf{x_i})\mathbf{p} - 2\mathbf{p}^T J(\mathbf{x_i})^T \Delta\mathbf{x_i} + \Delta\mathbf{x_i}^T \Delta\mathbf{x_i} && (5.39)\\
&= \mathbf{p}^T \left(\sum_i J(\mathbf{x_i})^T J(\mathbf{x_i})\right)\mathbf{p} - 2\mathbf{p}^T \left(\sum_i J(\mathbf{x_i})^T \Delta\mathbf{x_i}\right) + \sum_i \Delta\mathbf{x_i}^T \Delta\mathbf{x_i} && (5.40)\\
&= \mathbf{p}^T \mathbf{A}^{6\times 6}\mathbf{p} - 2\mathbf{p}^T \mathbf{b}^{6\times 1} + c && (5.41)\\
&= \begin{bmatrix} ? & ? & ? & ? & ? & ? \end{bmatrix} \begin{bmatrix} - & - & - & - & - & - \\ - & - & - & - & - & - \\ - & - & - & - & - & - \\ - & - & - & - & - & - \\ - & - & - & - & - & - \\ - & - & - & - & - & - \end{bmatrix} \begin{bmatrix} ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} && (5.42)\\
&\quad -2 \begin{bmatrix} - & - & - & - & - & - \end{bmatrix} \begin{bmatrix} ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} + \begin{bmatrix} - \end{bmatrix} && (5.43)
\end{aligned}
$$

The matrix $\mathbf{A}$ is called the Hessian and is given by the following form:

$$\mathbf{A} = \sum_i J(\mathbf{x_i})^T J(\mathbf{x_i}) \tag{5.44}$$

$$= \sum_i \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ x_i & 0 \\ y_i & 0 \\ 0 & x_i \\ 0 & y_i \end{bmatrix} \begin{bmatrix} 1 & 0 & x_i & y_i & 0 & 0 \\ 0 & 1 & 0 & 0 & x_i & y_i \end{bmatrix} \tag{5.45}$$

$$= \sum_i \begin{bmatrix} 1 & 0 & x_i & y_i & 0 & 0 \\ 0 & 1 & 0 & 0 & x_i & y_i \\ x_i & 0 & x_i^2 & x_i y_i & 0 & 0 \\ y_i & 0 & x_i y_i & y_i^2 & 0 & 0 \\ 0 & x_i & 0 & 0 & x_i^2 & x_i y_i \\ 0 & y_i & 0 & 0 & x_i y_i & y_i^2 \end{bmatrix} \tag{5.46}$$

$$= \begin{bmatrix} N & 0 & \sum_i x_i & \sum_i y_i & 0 & 0 \\ 0 & 1 & 0 & 0 & \sum_i x_i & \sum_i y_i \\ \sum_i x_i & 0 & \sum_i x_i^2 & \sum_i x_i y_i & 0 & 0 \\ \sum_i y_i & 0 & \sum_i x_i y_i & \sum_i y_i^2 & 0 & 0 \\ 0 & \sum_i x_i & 0 & 0 & \sum_i x_i^2 & \sum_i x_i y_i \\ 0 & \sum_i y_i & 0 & 0 & \sum_i x_i y_i & \sum_i y_i^2 \end{bmatrix} \tag{5.47}$$

$$\mathbf{b} = \sum_i J(\mathbf{x_i})^T \Delta \mathbf{x_i} \tag{5.48}$$

$$= \sum_i \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ x_i & 0 \\ y_i & 0 \\ 0 & x_i \\ 0 & y_i \end{bmatrix} \begin{bmatrix} \Delta x_i \\ \Delta y_i \end{bmatrix} \tag{5.49}$$

$$= \begin{bmatrix} \sum_i \Delta x_i \\ \sum_i \Delta y_i \\ \sum_i x_i \Delta x_i \\ \sum_i y_i \Delta x_i \\ \sum_i x_i \Delta y_i \\ \sum_i y_i \Delta y_i \end{bmatrix} \tag{5.50}$$

The above Hessian, $\mathbf{A}$, and the vector $\mathbf{b}$ can be computed entirely using matrix operations as follows:

Let the corresponding pairs of points be stored the two matrices as shown below. The corresponding columns in the two matrices store the corresponding pairs of points.

$$\mathbf{X} = \begin{bmatrix} x_1 & x_2 & \cdots & x_N \\ y_1 & y_2 & \cdots & y_N \\ 1 & 1 & \cdots & 1 \end{bmatrix}, \mathbf{X}' = \begin{bmatrix} x'_1 & x'_2 & \cdots & x'_N \\ y'_1 & y'_2 & \cdots & y'_N \\ 1 & 1 & \cdots & 1 \end{bmatrix} \tag{5.51}$$

The displacement between the corresponding points are stored in the following matrix. Note we ignore the third row of ones.

$$\Delta\mathbf{Z}^{2\times N} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} (\mathbf{X}' - \mathbf{X}) \tag{5.52}$$

$$\Delta\mathbf{X}^{2N\times 1} = \texttt{row scan into vector}(\Delta\mathbf{Z}) \tag{5.53}$$

Two constant matrices below are used to select coordinates from the point above to create the Jacobian

$$\mathbf{P_1} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{P_2} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.54}$$

$$\mathbf{J_1} = \mathbf{X}^T\mathbf{P_1} = \begin{bmatrix} 1 & 0 & x_1 & y_1 & 0 & 0 \\ 1 & 0 & x_2 & y_2 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & x_N & y_N & 0 & 0 \end{bmatrix} \tag{5.55}$$

$$\mathbf{J_2} = \mathbf{X}^T\mathbf{P_2} = \begin{bmatrix} 0 & 1 & 0 & 0 & x_1 & y_1 \\ 0 & 1 & 0 & 0 & x_2 & y_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & 0 & 0 & x_N & y_N \end{bmatrix} \tag{5.56}$$

$$\mathbf{J}^{2N\times 6} = \begin{bmatrix} \mathbf{J_1}^{N\times 6} \\ \mathbf{J_2}^{N\times 6} \end{bmatrix} \tag{5.57}$$

$$\mathbf{A}^{6\times 6} = (\mathbf{J}^T)^{6\times 2N}(\mathbf{J})^{2N\times 6} \tag{5.58}$$

$$\mathbf{b}^{6\times 1} = (\mathbf{J}^T)^{6\times 2N}(\Delta\mathbf{X})^{2N\times 1} \tag{5.59}$$

To find the minimum of the total residual, we take the derivative with respect to the parameters and set it to zero.

$$\frac{\partial r}{\partial \mathbf{p}} = 0 \tag{5.60}$$

$$\mathbf{Ap} = \mathbf{b} \tag{5.61}$$

$$\mathbf{p} = \mathbf{A}^{-1}\ \mathbf{b} \tag{5.62}$$

When does the inverse exist? What is the minimum number of point correspondences do we need?

Instead of inverse (or pseudo inverse), solving this via QR decomposition is more stable. Pseudo inverse is numerically unstable if the lowest eigenvalue is close to zero, i.e. ill conditioned. This could be the case if points are clustered together, i.e. not spread out across the region of interest. This is less likely to happen in the current case of computing affine transformations between point sets, but can happen in other settings where we need to compute the linear least squares estimate. Hence, the solving via QR decomposition is something that you need to be know and keep in mind as an option.

$$\mathbf{Ap} = \mathbf{b} \tag{5.63}$$

$$\mathbf{QRp} = \mathbf{b} \tag{5.64}$$

$$\mathbf{Rp} = \mathbf{Q}^T\mathbf{b} \tag{5.65}$$

where $\mathbf{Q}$ is a orthonormal matrix, $\mathbf{QQ}^T = \mathbf{I}$ and $\mathbf{R}$ is an upper triangular matrix. The solution for $\mathbf{p}$ is constructed by working the upper triangular matrix from the last row upwards towards the first row.

QR decomposition is the basis of more complex algorithms, such as computing the inverse, SVD, and eigenvalue decompositions. Unlike the SVD and eigenvalue decompositions, QR factorization does not require iteration and can be computed exactly in $)(MN^2 + N^3)$ operations, for a $M \times N$ matrix.

5.2 Jacobian for Similarity Transformation Estimation

Like the affine transformation, the similarity transformation is linear in its parameters. As a result, the single step estimation approach for affine estimation will work. The only difference will be in the Jacobian and consequently the Hessian matrices. The parameter vector is just four dimensional instead of six for affine transformation, i.e. $\mathbf{p} = \begin{bmatrix} t_x & t_y & a & b \end{bmatrix}^T$.

The Jacobian, corresponding to the i-the point, $\mathbf{J}(\mathbf{x_i})$ is given by

$$\mathbf{J}(\mathbf{x_i}) = \begin{bmatrix} 1 & 0 & x_i & y_i \\ 0 & 1 & y_i & x_i \end{bmatrix} \tag{5.66}$$

In terms of implementation, the only difference will be in the choice of constant matrices used to select coordinates from the point above to create the Jacobian (Eq. 5.54). They would following for similarity transformation.

$$\mathbf{P_1} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \mathbf{P_2} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{5.67}$$

5.3 Jacobian for 2D general rotation alignment

In general, the residual is not a linear function of the rotation parameter, θ.

$$\mathbf{r}_i = f(\widetilde{\mathbf{x}}_\mathbf{i}; \mathbf{p}) - \widetilde{\mathbf{x}}_\mathbf{i}' \tag{5.68}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} - \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} \tag{5.69}$$

$$\tag{5.70}$$

The parameter vector is three dimensional and is given by $\mathbf{p} = \begin{bmatrix} t_x & t_y & \theta \end{bmatrix}^T$. The Jacobian, corresponding to the i-the point, $\mathbf{J}(\mathbf{x_i}; \mathbf{p})$ is given by

$$\mathbf{J}(\mathbf{x_i};\mathbf{p}) = \frac{\partial}{\partial \mathbf{p}} f(\widetilde{\mathbf{x}}_\mathbf{i};\mathbf{p}) \tag{5.71}$$

$$= \begin{bmatrix} \frac{\partial}{\partial t_x} f(\widetilde{\mathbf{x}}_\mathbf{i};\mathbf{p}) & \frac{\partial}{\partial t_y} f(\widetilde{\mathbf{x}}_\mathbf{i};\mathbf{p}) & \frac{\partial}{\partial \theta} f(\widetilde{\mathbf{x}}_\mathbf{i};\mathbf{p}) \end{bmatrix} \tag{5.72}$$

$$= \begin{bmatrix} 1 & 0 & (-\sin(\theta)x_i - \cos(\theta)y_i) \\ 0 & 1 & (\cos(\theta)x_i - \sin(\theta)y_i) \end{bmatrix} \tag{5.73}$$

Note that the Jacobian is a function of the parameter, θ and the derivative of the residual (Eq. 5.60) will not yield a linear equation from which we can derive a closed form solution. Unlike for the affine case, there is no linear inversion process to estimate the parameters.

5.4 Jacobian for 2D Homography

Recall that the transformation can be represented as

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.74}$$

$$\widetilde{\mathbf{x}}' = \widetilde{\mathbf{H}}\widetilde{\mathbf{x}} \tag{5.75}$$

$$\mathbf{x}' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a/c \\ b/c \end{bmatrix} \tag{5.76}$$

Note that scaling of the transformation matrix, $\widetilde{\mathbf{H}}$, results in the same point $\mathbf{x}'$. This means that the transformation matrix can be known only up to a scale. Another aspect of this is that there are infinite number of solutions. To constrain the solution space, we can one of the parameters of the transformation matrix to a constant value. Typically, this is h_{22}, which is set to 1. This, along with reparameterization of the transformation so that $\mathbf{p} = \mathbf{0}$ is the identity transformation, we arrive at the following effective representation of the transformation.

$$\begin{bmatrix} a \\ b \\ D \end{bmatrix} = \begin{bmatrix} 1+h_{00} & h_{01} & h_{02} \\ h_{10} & 1+h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{5.77}$$

$$\hat{\mathbf{x}}' = \begin{bmatrix} \hat{x}'_i \\ \hat{y}'_i \end{bmatrix} = \begin{bmatrix} a/D \\ b/D \end{bmatrix} \tag{5.78}$$

The parameter vector is eight dimensional and is given by $\mathbf{p} = \begin{bmatrix} h_{00} & h_{01} & \cdots & h_{21} \end{bmatrix}^T$. The Jacobian, corresponding to the i-the point, $\mathbf{J}(\mathbf{x_i}; \mathbf{p})$ is given by

$$\mathbf{J}(\mathbf{x_i}; \mathbf{p}) = \frac{\partial}{\partial \mathbf{p}} f(\widetilde{\mathbf{x}}_\mathbf{i}; \mathbf{p}) \tag{5.79}$$

$$= \begin{bmatrix} \frac{\partial}{\partial h_{00}} \hat{x}'_i & \frac{\partial}{\partial h_{01}} \hat{x}'_i & \cdots & \frac{\partial}{\partial h_{21}} \hat{x}'_i \\ \frac{\partial}{\partial h_{00}} \hat{y}'_i & \frac{\partial}{\partial h_{01}} \hat{y}'_i & \cdots & \frac{\partial}{\partial h_{21}} \hat{y}'_i \end{bmatrix} \tag{5.80}$$

$$\hat{x}'_i = \frac{(1+h_{00})x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + 1} \tag{5.81}$$

$$\hat{y}'_i = \frac{(1+h_{10})x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + 1} \tag{5.82}$$

$$\tag{5.83}$$

Note $\frac{\partial \hat{x}'_i}{\partial h_{10}} = \frac{\partial \hat{x}'_i}{\partial h_{11}} = \frac{\partial \hat{x}'_i}{\partial h_{12}} = 0$, as $\hat{x}'_i$ does not depend on these parameters. Similarly, $\frac{\partial \hat{y}'_i}{\partial h_{00}} = \frac{\partial \hat{y}'_i}{\partial h_{01}} = \frac{\partial \hat{y}'_i}{\partial h_{02}} = 0$.

For ease of notation, let us use D to represent the denominator, $h_{20}x_i + h_{21}y_i + 1$. Then, $\frac{\partial \hat{x}'_i}{\partial h_{00}} = \frac{x_i}{D}$, $\frac{\partial \hat{x}'_i}{\partial h_{01}} = \frac{y_i}{D}$, and $\frac{\partial \hat{x}'_i}{\partial h_{02}} = \frac{1}{D}$.

And,

$$\frac{\partial \hat{x}'_i}{\partial h_{20}} = -\frac{x_i}{D^2}((1+h_{00})x_i + h_{01}y_i + h_{02}) \tag{5.84}$$

$$= -\frac{x_i}{D}\frac{(1+h_{00})x_i + h_{01}y_i + h_{02}}{D} \tag{5.85}$$

$$= -\frac{x_i \hat{x}'_i}{D} \tag{5.86}$$

$$\frac{\partial \hat{x}'_i}{\partial h_{21}} = -\frac{y_i}{D^2}((1+h_{00})x_i + h_{01}y_i + h_{02}) \quad (5.87)$$

$$= -\frac{y_i}{D}\frac{(1+h_{00})x_i + h_{01}y_i + h_{02}}{D} \quad (5.88)$$

$$= -\frac{y_i \hat{x}'_i}{D} \quad (5.89)$$

The Jacobian for projective transformation (homography) is given by

$$J(\mathbf{x_i}; \mathbf{p}) = \frac{1}{D}\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i\hat{x}'_i & -y_i\hat{x}'_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -x_i\hat{y}'_i & -y_i\hat{y}'_i \end{bmatrix} \quad (5.90)$$

Note that the $(\hat{x}'_i, \hat{y}'_i)$ and the denominator $D = h_{20}x_i + h_{21}y_i + 1$, depend on the current parameter settings.

Also note that the Jacobian involves predicted feature locations based on the current model parameters, $(\hat{x}'_i, \hat{y}'_i)$, instead of sensed feature locations in the next image, (x'_i, y'_i).

5.5 Non-linear Least Squares: Gauss–Newton approximation

We adopt an iterative process, where at each step update with a linear estimate. Starting with an initial rough estimate $\mathbf{p_0}$, which could be $\mathbf{p_0} = \mathbf{0}$ we iteratively add increments, $\Delta\mathbf{p}$ to the parameters, such that the residual is reduced to the maximum extent.

$$E_{NLS}(\Delta\mathbf{p}) = \sum_i \|f(\widetilde{\mathbf{x}}_\mathbf{i}; \mathbf{p} + \Delta\mathbf{p}) - \widetilde{\mathbf{x}}_\mathbf{i}'\|^2 \quad (5.91)$$

$$(5.92)$$

At each iteration $\mathbf{p}$ is known but $\Delta\mathbf{p}$ is unknown.

Using Taylor series, the function $f(.)$ at $\mathbf{p} + \Delta\mathbf{p}$ can be approximated using the Jacobian, $J(\mathbf{x_i}; \mathbf{p})$, as

$$f(\widetilde{\mathbf{x}}_\mathbf{i}; \mathbf{p} + \Delta\mathbf{p}) \approx f(\widetilde{\mathbf{x}}_\mathbf{i}; \mathbf{p}) + \frac{\partial}{\partial \mathbf{p}} f(\widetilde{\mathbf{x}}_\mathbf{i}; \mathbf{p})\Delta\mathbf{p} \quad (5.93)$$

$$= f(\widetilde{\mathbf{x}}_\mathbf{i}; \mathbf{p}) + J(\mathbf{x_i}; \mathbf{p})\Delta\mathbf{p} \quad (5.94)$$

Note that the Jacobian, in general, could be a function of the parameters along with $\mathbf{x_i}$, so the notation includes $\mathbf{p}$. Only for linear transformation is the Jacobian just a function of $\mathbf{x_i}$. Like what we saw for affine transform. Using this the error can be *approximated* by

$$E_{NLS}(\Delta\mathbf{p}) = \sum_i \|J(\mathbf{x_i};\mathbf{p})\Delta\mathbf{p} - \left(\widetilde{\mathbf{x}_\mathbf{i}}' - f(\widetilde{\mathbf{x}_\mathbf{i}};\mathbf{p})\right)\|^2 \quad (5.95)$$

$$= \sum_i \|J(\mathbf{x_i};\mathbf{p})\Delta\mathbf{p} - \Delta\mathbf{r_i}\|^2 \quad (5.96)$$

Compare this with the expression of error minimized for linear least squares (Eq. 5.32).

$$E_{NLS}(\Delta\mathbf{p}) = \sum_i \left(J(\mathbf{x_i};\mathbf{p})\Delta\mathbf{p} - \Delta\mathbf{r_i}\right)^T \left(J(\mathbf{x_i};\mathbf{p})\Delta\mathbf{p} - \Delta\mathbf{r_i}\right) \quad (5.97)$$

$$= \Delta\mathbf{p}^T \left(\sum_i J(\mathbf{x_i};\mathbf{p})^T J(\mathbf{x_i};\mathbf{p})\right)\Delta\mathbf{p} \quad (5.98)$$

$$-2\Delta\mathbf{p}^T \left(\sum_i J(\mathbf{x_i};\mathbf{p})^T \Delta\mathbf{r_i}\right) \quad (5.99)$$

$$+\sum_i \Delta\mathbf{r_i}^T \Delta\mathbf{r_i} \quad (5.100)$$

$$= \Delta\mathbf{p}^T\mathbf{A}\Delta\mathbf{p} - 2\Delta\mathbf{p}^T\mathbf{b} + \mathbf{c} \quad (5.101)$$

$$(5.102)$$

To minimize $E_{NLS}(\Delta\mathbf{p})$ we set the partial derivative to zero.

$$\frac{\partial}{\partial\Delta\mathbf{p}} E_{NLS}(\Delta\mathbf{p}) = 0 \quad (5.103)$$

$$\mathbf{A}\Delta\mathbf{p} = \mathbf{b} \quad (5.104)$$

$$\Delta\mathbf{p} = \mathbf{A}^{-1}\mathbf{b} \quad (5.105)$$

The overall approach is show in Algorithm 1.

Algorithm 1 Non-linear Least Square Transformation Estimator

procedure NONLINEAR($\{(\mathbf{x_i}, \mathbf{x_i}')|i = 1, \cdots, N\}$)
▷ Input are corresponding points.
$\begin{bmatrix}\hat{t}_x & \hat{t}_y & \hat{a}_{00} & \hat{a}_{01} & \hat{a}_{10} & \hat{a}_{11}\end{bmatrix} \leftarrow$ LINEAR($\{(\mathbf{x_i}, \mathbf{x_i}')|i = 1, \cdots, N\}$)
▷ Best fitting affine transformation.
$\hat{\mathbf{p}_0}^T = \begin{bmatrix}\hat{t}_x & \hat{t}_y & \hat{a}_{00} & \hat{a}_{01} & \hat{a}_{10} & \hat{a}_{11} & 0 & 0\end{bmatrix}$.
▷ Initial estimate of parameters.
$k \leftarrow 0$ ▷ Iteration counter.
$\Delta\mathbf{r_i} \leftarrow \widetilde{\mathbf{x}_\mathbf{i}}' - f(\widetilde{\mathbf{x}_\mathbf{i}}; \mathbf{p_0})$, for $i = 1, \cdots, N$.
$r \leftarrow \sum_i \|\Delta\mathbf{r_i}\|$ ▷ Total sum of squares residual.
while $r > \epsilon$ **do**
$\mathbf{A} \leftarrow \sum_i J(\mathbf{x_i}; \mathbf{p_k})^T J(\mathbf{x_i}; \mathbf{p_k})$
$\mathbf{b} \leftarrow \sum_i J(\mathbf{x_i}; \mathbf{p_k})^T \Delta\mathbf{r_i}$
$\Delta\mathbf{p_k} \leftarrow \mathbf{A}^{-1}\mathbf{b}$ ▷ Could be done using QR decomposition.
$\mathbf{p_{k+1}} \leftarrow \mathbf{p_k} + \Delta\mathbf{p_k}$
$\Delta\mathbf{r_i} \leftarrow \widetilde{\mathbf{x}_\mathbf{i}}' - f(\widetilde{\mathbf{x}_\mathbf{i}}; \mathbf{p_k})$, for $i = 1, \cdots, N$.
$r \leftarrow \sum_i \|\Delta\mathbf{r_i}\|$ ▷ Total sum of squares residual.
$k \leftarrow k + 1$
return $\mathbf{p_k}, r$ ▷ Return best fitting parameters and residual.

Chapter 6

Perspective Camera Model

- $\tilde{\mathbf{p}}$ is a 4 by 1 vector, $\begin{bmatrix} X_p & Y_p & Z_p & 1 \end{bmatrix}^T$ representing the homogeneous coordinates with respect to the world coordinates.
- $\tilde{\mathbf{p}}_c$ is a 4 by 1 vector, $\begin{bmatrix} X_c & Y_c & Z_c & 1 \end{bmatrix}^T$ representing the homogeneous coordinates with respect to the camera coordinates.
- $\tilde{\mathbf{x}}_c$ is a 3 by 1 vector, $\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T$ representing the homogeneous coordinates of the perspective projection of the 3D point onto the image with respect to the camera coordinates. See Figure 2.9 of textbook (page 52 of pdf file).
- $\mathbf{x}_c$ is non-homogeneous coordinate corresponding to the above homogeneous coordinate $\mathbf{x}_c = \begin{bmatrix} \frac{x_1}{x_3} & \frac{x_2}{x_3} \end{bmatrix}^T$.
- $\hat{\mathbf{x}}_c$ is a 2 by 1 vector, $\begin{bmatrix} \hat{x}_c & \hat{y}_c \end{bmatrix}^T$ representing the image location after lens distortion.
- $\mathbf{x}_s$ is a 2 by 1 vector, $\begin{bmatrix} x_s & y_s \end{bmatrix}^T$ representing the image location in pixel coordinates, with respect to the rows and columns of the image array.

1. What is the relationship between the world coordinates, $\tilde{\mathbf{p}}$, and the camera coordinates $\tilde{\mathbf{p}}_c$? It is a rigid 3D rotation and translation relationship.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_p \\ Y_p \\ Z_p \\ 1 \end{bmatrix} \tag{6.1}$$

2. What is the relationship between $\tilde{\mathbf{p}}_c$ and $\tilde{\mathbf{x}}_c$? The focal length is denoted by f.

   Consider a pinhole camera model with forward projection. The image plane is at $Z_c = f$. A ray through the image pixel (x_c, y_c) is denoted by the vector $\begin{bmatrix} x_c & y_c & f \end{bmatrix}^T$. This ray should pass through $\tilde{\mathbf{x}}_c$. Mathematically speaking, there exists some constant, k, such that

$$\begin{bmatrix} x_c \\ y_c \\ f \end{bmatrix} = k \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{6.2}$$

   We can eliminate k using the equality of the last component, $k = \frac{f}{Z_c}$. Using this, we can write.

$$x_c = f\frac{X_c}{Z_c} \tag{6.3}$$

$$y_c = f\frac{Y_c}{Z_c} \tag{6.4}$$

   In homogeneous coordinates, this can be written as:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{6.5}$$

$$\begin{bmatrix} x_c \\ y_c \end{bmatrix} = \begin{bmatrix} x_1/x_3 \\ x_2/x_3 \end{bmatrix} \tag{6.6}$$

3. What is the relationship between lens distorted coordinates $\hat{\mathbf{x}}_c$ and (x_c, y_c)? See Fig. 2.13 on page 59 of pdf for examples of lens distortions. Mathematically, the simplest radial distortion models can be expressed using low-order polynomials,

$$\begin{bmatrix} \hat{x}_c \\ \hat{y}_c \end{bmatrix} = \begin{bmatrix} x_c(1 + \kappa_1 r^2 + \kappa_2 r^4) \\ y_c(1 + \kappa_1 r^2 + \kappa_2 r^4) \end{bmatrix} \tag{6.7}$$

4. What is the relationship between the image pixel coordinates, $\mathbf{x}_s$, and $\hat{\mathbf{x}}_c$? The pixel aspect ratio is denoted by a, and the optical center is denoted by (c_x, c_y), in pixel coordinates.

$$\begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & c_x \\ 0 & -a & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}_c \\ \hat{y}_c \\ 1 \end{bmatrix} \tag{6.8}$$

   The signs can be used to take into account the axes handedness as needed.

Without the lens distortion, (item 3 above), the transformation of the 3D point to the image point (in homogeneous coordinates) is given as a product of a 3 by 3 intrinsic camera parameter $\mathbf{K}$ and a 3 by 4 extrinsic parameter matrix composed of a 3 by 3 rotation matrix, $\mathbf{R}$, and a 3 by 1 translation vector $\mathbf{t}$.

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -f & 0 & c_x & 0 \\ 0 & -af & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_p \\ Y_p \\ Z_p \\ 1 \end{bmatrix} \tag{6.9}$$

$$\tilde{\mathbf{x}}_c = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \tilde{\mathbf{p}} \tag{6.10}$$

6.1 Points on a 2D plane in 3D

Consider points on the plane $Z_p = 0$. The image points will be related to coordinates on the plane using a homography as follows.

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -f & 0 & c_x & 0 \\ 0 & -af & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & t_x \\ r_{10} & r_{11} & t_y \\ r_{20} & r_{21} & t_z \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_p \\ Y_p \\ 1 \end{bmatrix} \tag{6.11}$$

$$= \begin{bmatrix} -f & 0 & c_x \\ 0 & -af & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & t_x \\ r_{10} & r_{11} & t_y \\ r_{20} & r_{21} & t_z \end{bmatrix} \begin{bmatrix} X_p \\ Y_p \\ 1 \end{bmatrix} \tag{6.12}$$

$$= \begin{bmatrix} - & - & - \\ - & - & - \\ - & - & - \end{bmatrix} \begin{bmatrix} X_p \\ Y_p \\ 1 \end{bmatrix} \tag{6.13}$$

$$\tilde{\mathbf{x}}_\mathbf{c} = \mathbf{H}^{3\times 3}\tilde{\mathbf{x}}_\mathbf{p} \tag{6.14}$$

Thus, points on a plane, whose coordinates are specified with respect to the plane are related to the image points using a homography.

Now let us consider two cameras looking a plane, which is a scenario we might have when we are surveilling events on a flat floor.

The homographies relating the image points for the individual cameras are given by

$$\tilde{\mathbf{x}}_{\mathbf{1c}} = \mathbf{H_1}\tilde{\mathbf{x}}_\mathbf{p} \tag{6.15}$$

$$\tilde{\mathbf{x}}_{\mathbf{2c}} = \mathbf{H_2}\tilde{\mathbf{x}}_\mathbf{p} \tag{6.16}$$

Note $\tilde{\mathbf{x}}_\mathbf{p} = \begin{bmatrix} X_p \\ Y_p \\ 1 \end{bmatrix}$

6.2 Points on one plane from multiple cameras

We can directly relate the image pixel in the two images directly, by inverting one of the equations and plugging into the other.

$$\tilde{\mathbf{x}}_\mathbf{p} = \mathbf{H_2}^{-1}\tilde{\mathbf{x}}_{\mathbf{2c}} \tag{6.17}$$

$$\tilde{\mathbf{x}}_{\mathbf{1c}} = \mathbf{H_1}\mathbf{H_2}^{-1}\tilde{\mathbf{x}}_{\mathbf{2c}} \tag{6.18}$$

Assuming the homographies are invertible, which will be case unless we are viewing the plane under degenerate conditions, the image points will be related through a linear homography.

6.3 Image of a 2D line

A line can be parameterized by a point on the line, $\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$, and the unit direction vector, $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$, along the line.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \lambda \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{6.19}$$

The parameter λ represents distance along the line ranges from $-\infty$ and ∞. If we restrict the value of λ to a range them we get a line segment.

The image of a point in this line will be given by

$$x_c = f\frac{x_0 + \lambda a}{z_0 + \lambda c} \tag{6.20}$$

$$y_c = f\frac{y_0 + \lambda b}{z_0 + \lambda c} \tag{6.21}$$

$$\tag{6.22}$$

When $\lambda \to \infty$, we get a point at infinity on this line. The image of this point at infinity will be given by

$$x_\infty = \lim_{\lambda\to\infty} f\frac{x_0/\lambda + a}{z_0/\lambda + c} = f\frac{a}{c} \tag{6.23}$$

$$y_\infty = \lim_{\lambda\to\infty} f\frac{y_0/\lambda + b}{z_0/\lambda + c} = f\frac{b}{c} \tag{6.24}$$

$$\tag{6.25}$$

6.4 Parallel Lines

All parallel lines will have the same direction, $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$, and hence the point at infinity for all of them will be the same, i.e. the image of parallel lines will appear to meet at this point. This point is called vanishing point.

Note that if we can detect the vanishing point in the image, we can infer the direction of the constituent parallel lines in 3D, with respect to the camera. This can be an useful quantity in, for example, robot navigation or automated driving.

6.5 Parallel lines on a plane

Consider multiple sets of parallel lines that line on parallel planes, such as lines on the roof and the floor of a room, or lines of opposite walls in a room. Let the surface normals of the parallel planes be denoted by $\pm \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix}$. The directions of the parallel lines will be perpendicular to the surface normal direction. Thus,

$$\begin{aligned} n_1 a + n_2 b + n_3 c &= 0 && (6.26) \\ n_1 \frac{a}{c} + n_2 \frac{b}{c} + n_3 &= 0 && (6.27) \\ n_1 f \frac{a}{c} + n_2 f \frac{b}{c} + f n_3 &= 0 && (6.28) \\ n_1 x_\infty + n_2 y_\infty + f n_3 &= 0 && (6.29) \end{aligned}$$

This means that the vanishing points (x_∞, y_∞) corresponding to the each set of parallel lines will all lie on a line. This line will be horizon if the camera is oriented vertically with respect to the roof and the floor, i.e., $n_1 = n_3 = 0$.

Chapter 7

2D to 3D Camera Calibration (Tsai)

Without the lens distortion, the transformation of the 3D point to the image point (in homogeneous coordinates) is given as a product of a 3 by 3 intrinsic camera parameter $\mathbf{K}$ and a 3 by 4 extrinsic parameter matrix composed of a 3 by 3 rotation matrix, $\mathbf{R}$, and a 3 by 1 translation vector $\mathbf{t}$.

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -f & 0 & c_x & 0 \\ 0 & -af & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_p \\ Y_p \\ Z_p \\ 1 \end{bmatrix} \tag{7.1}$$

$$.\tilde{\mathbf{x}}_c = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \tilde{\mathbf{p}} \tag{7.2}$$

The task of calibration is to be able to derive the intrinsic $\mathbf{K}$ and extrinsic parameters, $\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}$, given pairs consisting of 3D points, with respect to the world, $\tilde{\mathbf{X}}_p$, and their imaged location in terms of row and column in the image, $\tilde{\mathbf{x}}_c$.

1. Image a target of known geometry. The target should have points on it that can easily imaged. They should be distributed across all three dimensions. The image of the calibration object should occupy the entire image. Sometimes, instead of the calibration object, we establish a coordinate axis in the 3D scene with respect to some real 3D point such as the corner of a room and choose 3D points from the scene, based on actual 3D dimensions.

2. Establish correspondences between 3D target points, with respect to the world coordinates, and their 2D image location in pixel coordinates.

3. Obtain estimates of as many parameters as possible using linear least-squares fitting methods, which are fast.

   a) Estimate rotation and parts of the translation vector using first form of DLT.

   b) Estimate the rest of the translation vector and focal length using the second form of the DLT.

4. Impose orthogonality of the rotation matrix estimates. This does not affect the final result, since these estimated parameter values are used only as starting values for the final optimization, which is the next step.

5. The rest of the parameters are obtained using a nonlinear optimization method that finds the best fit between the observed image points and those predicted from the target model. Note that the DLT does not minimize the error in the image projection, but some other unknown quantity that led to convenient, linear equations. The resulting rotation matrix and translation components are not especially accurate as a result. This is acceptable only because we use the recovered values merely as estimates in the full non-linear optimization using the iterative method described in the textbook and the Tsai paper.

7.1 First form of Direct Linear Transform (DLT)

1. What is the ratio of $x_s - c_x$ and $y_s - c_y$? Assume that we have a good initial estimate of the optic center, (c_x, c_y). Usually, the center of the image is a good estimate.

$$\frac{x_s - c_x}{y_s - c_y} = \tag{7.3}$$

2. Rewrite the above in the following form.

$$(\qquad)r_{00} + (\qquad)r_{01} + (\qquad)r_{02} + \tag{7.4}$$
$$(\qquad)ar_{10} + (\qquad)ar_{11} + (\qquad)ar_{12} + \tag{7.5}$$
$$(\qquad)r_{10} + (\qquad)r_{11} + (\qquad)r_{12} + \tag{7.6}$$
$$(\qquad)t_x + (\qquad)t_y + (\qquad)t_z = 0 \tag{7.7}$$

We get one such equation for each 2D-3D point pair. The unknown (eight) parameter vector is given by

$$\begin{bmatrix} r_{00} & r_{01} & r_{02} & ar_{10} & ar_{11} & ar_{12} & t_x & at_y \end{bmatrix}^T \tag{7.8}$$

There is an unknown scale factor. If we have a solution for the eight unknowns, then any multiple of that solution is also a solution. In order to obtain an unique solution, we can arbitrarily set one unknown, t_x say — to one. (Assumes some relationship among the origins of the camera and the world. What is it?) So, we end up with 7 unknowns. So we need seven pairs of 2D-3D point correspondences.

Note the third row of the rotation matrix is not being estimated here. Hang on, we have a trick up our sleeves. If we can estimate the first two rows of the rotation matrix, we can infer the third from the orthogonality constraint. More on that a bit later.

Let $y' = y_s - c_y$ and $x' = x_s - c_x$.

$$\begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_w y' & Y_w y' & Z_w y' & y' & -X_w x' & -Y_w x' & -Z_w x' & -x' \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} r_{00} \\ r_{01} \\ r_{02} \\ ar_{10} \\ ar_{11} \\ ar_{12} \\ 1 \\ at_y \end{bmatrix} = 0 \tag{7.9}$$

The above can be put in the following form by moving the seven column, that multiply with $t_x = 1$, to the right as vector $\mathbf{b}$.

$$\begin{aligned} \mathbf{Ap} &= \mathbf{b} && (7.10)\\ \mathbf{A}^T\mathbf{Ap} &= \mathbf{A}^T\mathbf{b} && (7.11)\\ \mathbf{U\Sigma Vp} &= \mathbf{A}^T\mathbf{b} && (7.12)\\ \mathbf{p} &= \mathbf{V}^T\Sigma^{-1}\mathbf{U}^T\mathbf{A}^T\mathbf{b} && (7.13) \end{aligned}$$

Can we always do this? Think about it.

From the estimated $\mathbf{p} = \begin{bmatrix} p_0 & p_1 & \cdots & p_6 \end{bmatrix}^T$ we can unravel the rotation and pixel aspect ratio (a) by enforcing the property of a rotation matrix. The first property that we will exploit is that the rows of the rotation matrix have unit length. Let is denote the rows of the rotation matrix by $\mathbf{r}_x^T$, $\mathbf{r}_y^T$, and $\mathbf{r}_z^T$.

$$\begin{aligned} \|\mathbf{r}_x^T\|^2 &= r_{00}^2 + r_{01}^2 + r_{02}^2 = 1 && (7.14)\\ \|\mathbf{r}_y^T\|^2 &= r_{10}^2 + r_{11}^2 + r_{12}^2 = 1 && (7.15)\\ & && (7.16) \end{aligned}$$

Let $c_1 = \sqrt{p_0^2 + p_1^2 + p_2^2}$ and $c_2 = \sqrt{p_3^2 + p_4^2 + p_5^2}$. Divide $\mathbf{p}$ by c_1. The first three entries of this normalized vector will give us the first row of the rotation matrix. Normalize the next three entries of $\mathbf{p}$ to get the second row of the rotation matrix.

$$\begin{aligned} r_{00} = \frac{p_0}{c_1}, r_{01} = \frac{p_1}{c_1}, r_{02} = \frac{p_2}{c_1} && (7.17)\\ r_{10} = \frac{p_3}{c_2}, r_{11} = \frac{p_4}{c_2}, r_{12} = \frac{p_5}{c_2} && (7.18)\\ a = \frac{c_2}{c_1} && (7.19)\\ && (7.20) \end{aligned}$$

The third row of the rotation matrix is derived by imposing the constraint that the rows of the rotation matrix are orthogonal. We can get such as vector by taking the cross product of the estimates of the first two rows.

$$\mathbf{r}_z = \mathbf{r}_x \times \mathbf{r}_y \quad (7.21)$$

Note that $\|\mathbf{r}_z\| = 1$ because the two other vectors have unit norm. The rotation matrix now satisfies all the orthogonality and normality constraints, except that the first rows are not guaranteed to be orthogonal. We can rotate the vectors $\mathbf{r}_x$ and $\mathbf{r}_y$ in the plane containing them until they are orthogonal. The operation that will accomplish is the following transformation. It is basically subtracting the dot product of the vectors from each other. If the vectors are orthogonal to begin with, the dot product will be zero and nothing will need to be changed.

$$\begin{aligned} \mathbf{r}_x &\leftarrow \mathbf{r}_x - 0.5(\mathbf{r}_x^T \mathbf{r}_y)\mathbf{r}_y && (7.22)\\ \mathbf{r}_y &\leftarrow \mathbf{r}_y - 0.5(\mathbf{r}_x^T \mathbf{r}_y)\mathbf{r}_x && (7.23)\\ & && (7.24) \end{aligned}$$

After this transformation, we need to re-normalize the vectors

$$\begin{aligned} \mathbf{r}_x &\leftarrow \mathbf{r}_x / \|\mathbf{r}_x\| && (7.25)\\ \mathbf{r}_y &\leftarrow \mathbf{r}_y / \|\mathbf{r}_y\| && (7.26)\\ & && (7.27) \end{aligned}$$

Apart of the rotation matrix we also have estimates of parts of the translation vector

$$t_x = \frac{p_6}{c_1}, t_y = \frac{1}{c_1}, t_z = ? \tag{7.28}$$

We form an initial estimate of t_z the focal length f, we use the second form of the DLT, where we ignore the lens distortion. This is considered in the next section. Note that the DLT in current section respects the lens distortion and is invariant to it as it uses the ratio of the $x-$ and y-coordinates.

7.2 Second form of the DLT

TBD

Chapter 8

Structure from Motion (SFM)

8.1 Triangulation - 3D residuals

- Assume calibrated cameras. Lens distortion has been removed. Use Fig 7.2 as illustration.
- We know that the homogeneous coordinate of an image pixel (x_j, y_j) is related the 3D coordinates, which is unknown, as per the following equations. We assume that we have estimates of the intrinsic and extrinsic parameters.

$$\begin{bmatrix} w_j x_j \\ w_j y_j \\ w_j \end{bmatrix} = \begin{bmatrix} -f & 0 & c_x \\ 0 & -af & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{r}_x^T & t_x \\ \mathbf{r}_y^T & t_y \\ \mathbf{r}_z^T & t_z \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} X_p \\ Y_p \\ Z_p \\ 1 \end{bmatrix} \tag{8.1}$$

- Let the image of the 3D point $\mathbf{p} = (X_p, Y_p, Z_p)$ be denoted by $\mathbf{x_j}$ in the j-th camera. The intrinsic and extrinsic parameters of the j-th camera are denoted by $\mathbf{K}_j$, $\mathbf{R}_j$, and $\mathbf{t}_j$,

$$\tilde{\mathbf{x}}_\mathbf{j} = \mathbf{K}_j \begin{bmatrix} \mathbf{R}_j & \mathbf{t}_j \end{bmatrix} \tilde{\mathbf{p}} \tag{8.2}$$

$$\mathbf{K}_j^{-1} \tilde{\mathbf{x}}_\mathbf{j} = \begin{bmatrix} \mathbf{R}_j & \mathbf{t}_j \end{bmatrix} \tilde{\mathbf{p}} \tag{8.3}$$

$$\mathbf{R}_j^T \mathbf{K}_j^{-1} \tilde{\mathbf{x}}_\mathbf{j} = \begin{bmatrix} \mathbf{I} & \mathbf{R}_j^T \mathbf{t}_j \end{bmatrix} \tilde{\mathbf{p}} \tag{8.4}$$

$$\mathbf{R}_j^T \mathbf{K}_j^{-1} \tilde{\mathbf{x}}_\mathbf{j} = \mathbf{p} + \mathbf{R}_j^T \mathbf{t}_j \tag{8.5}$$

- $\mathbf{c}_j = -\mathbf{R}_j^T \mathbf{t}_j$ is the vector from the world coordinate origin to the camera coordinate origin, specified with respect to the *world* coordinates. Note $\mathbf{t}_j$ is the vector pointing the other way, i.e. from camera origin to world origin, but specified with respect to the *camera* coordinates.

- Using the above, we can rewrite the world coordinates as

$$\mathbf{p} = \mathbf{R}_j^T \mathbf{K}_j^{-1} \tilde{\mathbf{x}}_\mathbf{j} + \mathbf{c}_j \tag{8.6}$$

  Please keep in mind that $\mathbf{p}$ is specified with respect to the world coordinates.

  It appears that we have a closed form solution for the 3D location of the image point just from one equation, but that is not so! In the above equation, we have to specify the homogeneous coordinates of the image pixel location, $\tilde{\mathbf{x}}_\mathbf{j} = \begin{bmatrix} w_j x_j & w_j y_j & w_j \end{bmatrix}^T$. We know (x_j, y_j), the pixel location, but we do not know w_j. The point could be anywhere along the vector that start from the image coordinate origin and go through the image pixel (x_j, y_j). Refer to Figure 7.2 in the textbook.

- Re-writing the above equation using a compact notation.

$$\mathbf{p} = \mathbf{R}_j^T \mathbf{K}_j^{-1} \begin{bmatrix} w_j x_j \\ w_j y_j \\ w_j \end{bmatrix} + \mathbf{c}_j \tag{8.7}$$

$$\mathbf{p} = w_j \mathbf{R}_j^T \mathbf{K}_j^{-1} \begin{bmatrix} x_j \\ y_j \\ 1 \end{bmatrix} + \mathbf{c}_j \tag{8.8}$$

$$\mathbf{p} = d_j \hat{\mathbf{v}}_j \ + \mathbf{c}_j \tag{8.9}$$

  where $\hat{\mathbf{v}}_j$ is the normalized unit vector, along $\mathbf{R}_j^T \mathbf{K}_j^{-1} \begin{bmatrix} x_j \\ y_j \\ 1 \end{bmatrix}$, representing the vector through the image pixel, with respect to the *world* coordinates. We denote it by

$$\hat{\mathbf{v}}_j = \mathcal{N}(\mathbf{R}_j^T \mathbf{K}_j^{-1} \mathbf{x}_j) \tag{8.10}$$

- The residual $\mathbf{r}_j$ is the given by

$$\mathbf{r}_j = d_j \hat{\mathbf{v}}_j - (\mathbf{p} - \mathbf{c}_j) \tag{8.11}$$

  This is the vector connecting the lines through the image pixel and the vector to the 3D world coordinate, with respect to the *camera* coordinates, which is $(\mathbf{p} - \mathbf{c}_j)$. Use Figure 7.2 to illustrate this.

- The total residual over N-cameras is given by

$$\sum_{j=1}^{N} \|\mathbf{r}_j\|^2 = \sum_{j=1}^{N} (d_j \hat{\mathbf{v}}_j - (\mathbf{p} - \mathbf{c}_j))^T (d_j \hat{\mathbf{v}}_j - (\mathbf{p} - \mathbf{c}_j)) \tag{8.12}$$

  There are (N+3) unknowns: distances with respect to the N cameras d_j's, and the three world coordinates in $\mathbf{p}$.

- To minimize, take derivative with respect to d_j and set it to zero.

$$\begin{aligned}
\frac{\partial}{\partial d_j} (d_j \hat{\mathbf{v}}_j + \mathbf{c}_j - \mathbf{p})^T (d_j \hat{\mathbf{v}}_j + \mathbf{c}_j - \mathbf{p}) &= 0 && (8.13)\\
2(\hat{\mathbf{v}}_j)^T (d_j \hat{\mathbf{v}}_j + \mathbf{c}_j - \mathbf{p}) &= 0 && (8.14)\\
d_j + \hat{\mathbf{v}}_j^T (\mathbf{c}_j - \mathbf{p}) &= 0 && (8.15)\\
d_j &= \hat{\mathbf{v}}_j^T (\mathbf{p} - \mathbf{c}_j) && (8.16)
\end{aligned}$$

  Use Figure 7.2 in the textbook to trace out the geometry of the above equation.

- Plugging this back into the residual equation, we have

$$\begin{aligned}
\sum_{j=1}^{N} \|\mathbf{r}_j\|^2 &= \sum_{j=1}^{N} \|(\hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T (\mathbf{p} - \mathbf{c}_j) - (\mathbf{p} - \mathbf{c}_j))\|^2 && (8.17)\\
&= \sum_{j=1}^{N} ((\mathbf{I} - \hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j))^T ((\mathbf{I} - \hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j)) && (8.18)\\
& && (8.19)
\end{aligned}$$

- Setting the derivative of the residual with respect to $\mathbf{p}$ to zero, we have

$$\sum_{j=1}^{N}(\mathbf{I}-\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)^T((\mathbf{I}-\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)(\mathbf{p}-\mathbf{c}_j)) = 0 \tag{8.20}$$

$$\sum_{j=1}^{N}(\mathbf{I}-\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)(\mathbf{I}-\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)(\mathbf{p}-\mathbf{c}_j) = 0 \tag{8.21}$$

$$\sum_{j=1}^{N}(\mathbf{I}-2\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T+\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)(\mathbf{p}-\mathbf{c}_j) = 0 \tag{8.22}$$

$$\sum_{j=1}^{N}(\mathbf{I}-\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)(\mathbf{p}-\mathbf{c}_j) = 0 \tag{8.23}$$

$$\tag{8.24}$$

$$\left(\sum_{j=1}^{N}(\mathbf{I}-\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)\right)\mathbf{p}-\sum_{j=1}^{N}(\mathbf{I}-\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)\mathbf{c}_j = 0 \tag{8.25}$$

- The least square estimate of the 3D location of the point is given by

$$\mathbf{p}=\left(\sum_{j=1}^{N}(\mathbf{I}-\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)\right)^{-1}\sum_{j=1}^{N}(\mathbf{I}-\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)\mathbf{c}_j \tag{8.26}$$

  This equation is valid only when we have more than one camera. For one camera, we get a degenerate solution where $\mathbf{p}=\mathbf{c}_j$, i.e. the point is located at the camera origin!

8.2 Triangulation - 2D residuals

- Minimize the 2D image residuals. This can produce significantly better estimates if more of the cameras are closer to the 3D points than others, i.e. the points are distributed in 3D space.

- $$\tilde{\mathbf{x}}_\mathbf{j}=\mathbf{K}_j\begin{bmatrix}\mathbf{R}_j & \mathbf{t}_j\end{bmatrix}\tilde{\mathbf{p}}=\mathbf{P}_j\tilde{\mathbf{p}}=\begin{bmatrix}p_{00}^j & p_{01}^j & p_{02}^j & p_{03}^j\\ p_{10}^j & p_{11}^j & p_{12}^j & p_{13}^j\\ p_{20}^j & p_{21}^j & p_{22}^j & p_{23}^j\end{bmatrix}\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix} \tag{8.27}$$

- Here the entries of the projection matrix are known and the unknown are the homogeneous coordinates of the 3D point.

$$x_j = f_x(\mathbf{P}^j; \mathbf{p}) = \frac{p^j_{00}X + p^j_{01}Y + p^j_{02}Z + p^j_{03}}{p^j_{20}X + p^j_{21}Y + p^j_{22}Z + p^j_{23}} \tag{8.28}$$

$$y_j = f_y(\mathbf{P}^j; \mathbf{p}) = \frac{p^j_{10}X + p^j_{11}Y + p^j_{12}Z + p^j_{13}}{p^j_{20}X + p^j_{21}Y + p^j_{22}Z + p^j_{23}} \tag{8.29}$$

- The unknown is the vector $\mathbf{p} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$. Let the estimate of this vector at iteration k be denoted by $\mathbf{p}(k)$. Then we want to find a change in this vector, denoted by $\Delta\mathbf{p}(k)$ such that the residual with $\mathbf{p}(k+1) = \mathbf{p}(k) + \Delta\mathbf{p}$ is minimized.

  The total image residual for N cameras is given by

$$E^{(k)}_{NLS}(\Delta\mathbf{p}) = \sum_{j=1}^{N} \|\mathbf{x}_j - \mathbf{f}(\mathbf{P}^j; \mathbf{p}(k) + \Delta\mathbf{p})\|^2 \tag{8.30}$$

$$\approx \sum_{j=1}^{N} \|\mathbf{x}_j - \mathbf{f}(\mathbf{P}^j; \mathbf{p}(k)) + \frac{\partial \mathbf{f}(\mathbf{P}^j; \mathbf{p}(k))}{\partial \mathbf{p}} \Delta\mathbf{p}\|^2 \tag{8.31}$$

$$\approx \sum_{j=1}^{N} \|\mathbf{J}(\mathbf{P}^j; \mathbf{p}(k))\Delta\mathbf{p} - \mathbf{r}_j(k)\|^2 \tag{8.32}$$

  where $\mathbf{J}$ is the Jacobian for the k-iteration and $\mathbf{r}_j(k)$is the residual at the k-iteration.

- The Jacobian is 2 by 3 and are given by the entries

$$\mathbf{J}^j = \begin{bmatrix} \frac{\partial f_x}{\partial X} & \frac{\partial f_x}{\partial Y} & \frac{\partial f_x}{\partial Z} \\ \frac{\partial f_y}{\partial X} & \frac{\partial f_y}{\partial Y} & \frac{\partial f_y}{\partial Z} \end{bmatrix} \tag{8.33}$$

  Let $D(k) = p^j_{20}X(k) + p^j_{21}Y(k) + p^j_{22}Z(k) + p^j_{23}$ be the denominator at the k-th iteration.

$$\frac{\partial f_x}{\partial X} = \frac{1}{D(k)}(p^j_{00} - p^j_{20}x_j) \tag{8.34}$$
$$\frac{\partial f_x}{\partial Y} = \frac{1}{D(k)}(p^j_{01} - p^j_{21}x_j) \tag{8.35}$$
$$\frac{\partial f_x}{\partial Z} = \frac{1}{D(k)}(p^j_{02} - p^j_{22}x_j) \tag{8.36}$$
$$\frac{\partial f_y}{\partial X} = \frac{1}{D(k)}(p^j_{10} - p^j_{20}y_j) \tag{8.37}$$
$$\frac{\partial f_y}{\partial Y} = \frac{1}{D(k)}(p^j_{11} - p^j_{21}y_j) \tag{8.38}$$
$$\frac{\partial f_y}{\partial Z} = \frac{1}{D(k)}(p^j_{12} - p^j_{22}y_j) \tag{8.39}$$
$$\tag{8.40}$$

- Solve the following linear equation

$$\mathbf{A}\Delta\mathbf{p} = \mathbf{b} \tag{8.41}$$

  where $\mathbf{A} = \sum_j (\mathbf{J}^j)^T(\mathbf{J}^j)$ and $\mathbf{b} = \sum_j (\mathbf{J}^j)^T(r_j(k))$.

- Unique solution exists only if the rank of $\mathbf{A}$ is more than 3. Rank $((\mathbf{J}^j)^T(\mathbf{J}^j)) = 2$. Rank $(\mathbf{A}) = 2N$, so need to have at least two cameras.

- $\mathbf{p}(k+1) = \mathbf{p}(k) + \Delta\mathbf{p}$

- Recompute Jacobian and then the next change in parameters.

- Repeat until change in residual is small.

- Initialize the process using an estimate based on DLT as follows.

- $$\begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ (p^j_{20}x_j - p^j_{00}) & (p^j_{21}x_j - p^j_{01}) & (p^j_{22}x_j - p^j_{02}) & (p^j_{23}x_j - p^j_{03}) \\ (p^j_{20}y_j - p^j_{00}) & (p^j_{21}y_j - p^j_{01}) & (p^j_{22}Y_j - p^j_{02}) & (p^j_{23}y_j - p^j_{03}) \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = 0 \tag{8.42}$$

- It is best if we keep the fourth components of homogeneous coordinate of the 3D point, W, as an unknown. Do SVD and pick the vector corresponding to the smallest singular value as the estimate of the homogeneous coordinate. Normalize by the last element to obtain an estimate of the coordinates.

- Need at least two cameras, 4 rows in the matrix above to estimate the 4 unknowns.

8.3 2-frame SFM

What happens when we do not know the location of the 3D coordinates *and* also do not know the 3D camera transformations, $\mathbf{R}_j$ and $\mathbf{t}_j$, of each of the cameras. Let us consider the simpler problem of just two cameras. Focus on Figure 7.3 in the textbook.

- For simplified mathematical notations and without the loss of generality, let the "world" coordinate be at set at the first camera, i.e., $\mathbf{c}_0 = \mathbf{0}$ and $\mathbf{R}_0 = \mathbf{I}$.

- $\hat{\mathbf{x}}_j = \mathbf{K}_j^{-1}\mathbf{x}_j$ is the ray from the camera center to the pixel and out to the world, with respect to the *local* camera coordinates. Note that we need to know the intrinsic camera parameters.

- Compare this with the vector we had seen earlier, $\hat{\mathbf{v}}_j = \mathcal{N}(\mathbf{R}_j^T\mathbf{K}_j^{-1}\mathbf{x}_j)$, which is the ray from the camera center to the pixel and out to the world, but with respect to the *world* camera coordinates.

- The three vectors from $\overline{\mathbf{c}_0\mathbf{c}_1}$, $\overline{\mathbf{c}_0\mathbf{p}}$, and $\overline{\mathbf{c}_1\mathbf{p}}$, all line on a plane. We are ignoring the fact that image locations are quantized in pixel units and the rays might not pass exactly through the same 3D point in practice.
- The rotation matrix $\mathbf{R}$ and the translation vector $\mathbf{t}$ takes coordinates with respect to the left camera ($\mathbf{c}_0$) and express it with respect to the second camera ($\mathbf{c}_1$).

$$\mathbf{p}_1 = \mathbf{R}\mathbf{p}_0 + \mathbf{t} \tag{8.43}$$
$$w_1\hat{\mathbf{x}}_1 = w_2\mathbf{R}\hat{\mathbf{x}}_0 + \mathbf{t} \tag{8.44}$$

Take a cross product with vector $\mathbf{t}$ of both sides. Note $\mathbf{t} \times \mathbf{t} = 0$.

$$w_1\mathbf{t} \times \hat{\mathbf{x}}_1 = w_2\mathbf{t} \times \mathbf{R}\hat{\mathbf{x}}_0 \tag{8.45}$$

Take dot product with $\hat{\mathbf{x}}_\mathbf{1}$. The left hand side will be zero.

$$\hat{\mathbf{x}}_\mathbf{1}^T \left[\mathbf{t} \times \mathbf{R}\hat{\mathbf{x}}_0\right] = 0 \tag{8.46}$$
$$\hat{\mathbf{x}}_\mathbf{1}^T \left[\mathbf{t} \times \begin{bmatrix}\mathbf{r}_x^T \\ \mathbf{r}_y^T \\ \mathbf{r}_z^T\end{bmatrix} \hat{\mathbf{x}}_0\right] = 0 \tag{8.47}$$
$$\hat{\mathbf{x}}_\mathbf{1}^T \left[\begin{bmatrix}t_x \\ t_y \\ t_z\end{bmatrix} \times \begin{bmatrix}\mathbf{r}_x^T\hat{\mathbf{x}}_0 \\ \mathbf{r}_y^T\hat{\mathbf{x}}_0 \\ \mathbf{r}_z^T\hat{\mathbf{x}}_0\end{bmatrix}\right] = 0 \tag{8.48}$$
$$\hat{\mathbf{x}}_\mathbf{1}^T \left[\begin{bmatrix}t_x \\ t_y \\ t_z\end{bmatrix} \times \begin{bmatrix}\mathbf{r}_x^T\hat{\mathbf{x}}_0 \\ \mathbf{r}_y^T\hat{\mathbf{x}}_0 \\ \mathbf{r}_z^T\hat{\mathbf{x}}_0\end{bmatrix}\right] = 0 \tag{8.49}$$
$$\hat{\mathbf{x}}_\mathbf{1}^T \begin{bmatrix}t_y(\mathbf{r}_z^T\hat{\mathbf{x}}_0) - t_z(\mathbf{r}_y^T\hat{\mathbf{x}}_0) \\ t_z(\mathbf{r}_x^T\hat{\mathbf{x}}_0) - t_x(\mathbf{r}_z^T\hat{\mathbf{x}}_0) \\ t_x(\mathbf{r}_y^T\hat{\mathbf{x}}_0) - t_y(\mathbf{r}_x^T\hat{\mathbf{x}}_0)\end{bmatrix} = 0 \tag{8.50}$$
$$\hat{\mathbf{x}}_\mathbf{1}^T \begin{bmatrix}t_y(\mathbf{r}_z^T) - t_z(\mathbf{r}_y^T) \\ t_z(\mathbf{r}_x^T) - t_x(\mathbf{r}_z^T) \\ t_x(\mathbf{r}_y^T) - t_y(\mathbf{r}_x^T)\end{bmatrix} \hat{\mathbf{x}}_0 = 0 \tag{8.51}$$
$$\hat{\mathbf{x}}_\mathbf{1}^T \begin{bmatrix}\mathbf{t} \times \mathbf{r}_1 & \mathbf{t} \times \mathbf{r}_2 & \mathbf{t} \times \mathbf{r}_3\end{bmatrix} \hat{\mathbf{x}}_0 = 0 \tag{8.52}$$
$$\hat{\mathbf{x}}_\mathbf{1}^T \left[\mathbf{t} \times \mathbf{R}\right] \hat{\mathbf{x}}_0 = 0 \tag{8.53}$$

- $\mathbf{E} = \mathbf{t} \times \mathbf{R}$ is called the essential matrix.

$$\begin{bmatrix} x_1 & y_1 & 1 \end{bmatrix} \begin{bmatrix} e_{00} & e_{01} & e_{02} \\ e_{10} & e_{11} & e_{12} \\ e_{20} & e_{21} & e_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} = 0 \tag{8.54}$$

- If we do not known the intrinsic camera parameters, $\mathbf{K}$, i.e. focal length and optic center, we can still derive a similar constraint with the pixel coordinates, $\mathbf{x}_j$, the matrix we end up with is the *fundamental* matrix and is related to the *essential* matrix as

$$\mathbf{F} = \mathbf{K}^{-T}\mathbf{E}\mathbf{K}^{-1} \tag{8.55}$$

  However we will not be able to unravel the translation and rotation matrices from the fundamental matrix as it contain camera parameters

- The equation above is the *epipolar constraint*. For a known point in one of cameras, the corresponding point has to lie on a straight line on the other image.

$$(e_{00}x_0+e_{01}y_0+e_{02})x_1+(e_{10}x_0+e_{11}y_0+e_{12})y_1+(e_{20}x_0+e_{21}y_0+e_{22}) = 0 \tag{8.56}$$

- Each pair of correspondences result in one constraining equation.

$$(x_0x_1)e_{00}+(y_0x_1)e_{01}+x_1e_{02}+(x_0y_1)e_{10}+(y_0y_1)e_{11}+y_1e_{12}+x_0e_{10}+y_0e_{11}+e_{12} = 0 \tag{8.57}$$

- Given N pairs of correspondences, we will have N linear equations to solve.

$$\begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ (x_0x_1) & (y_0x_1) & x_1 & (x_0y_1) & (y_0y_1) & y_1 & x_0 & y_0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} e_{00} \\ e_{01} \\ e_{02} \\ e_{10} \\ e_{11} \\ e_{12} \\ e_{20} \\ e_{21} \\ e_{22} \end{bmatrix} = 0 \tag{8.58}$$

- The above equation can be solved by eigenvalues.

$$\mathbf{A}^{N\times 9}\mathbf{e}^{9\times 1} = 0 \tag{8.59}$$
$$\min\|\mathbf{Ae}\|^2 \tag{8.60}$$
$$\min(\mathbf{Ae})^T\mathbf{Ae} \tag{8.61}$$
$$\min \mathbf{e}^T(\mathbf{A}^T\mathbf{A})^{9\times 9}\mathbf{e} \tag{8.62}$$

  To avoid the degenerate solution, $\mathbf{e} = 0$ we add the constraint that $\|\mathbf{e}\| = 1$. Note that we can recover the vector $\mathbf{e}$ only upto a scale.

  So the minimization problem, using Lagrange multipliers, can be expressed as

$$\min \mathbf{e}^T(\mathbf{A}^T\mathbf{A})\mathbf{e} + \lambda(1 - \mathbf{e}^T\mathbf{e}) \tag{8.63}$$

  Taking the derivative of the above with respect to $\mathbf{e}$ and setting it to zero, we have

$$(\mathbf{A}^T\mathbf{A})\mathbf{e} = \lambda\mathbf{e} \tag{8.64}$$

- Solution is the eigenvector corresponding to the smallest eigenvalue of $\mathbf{A}^T\mathbf{A}$.
- Rearrange $\mathbf{e}$ into a 3 by 3 matrix $\mathbf{E}$
- Note the columns of $\mathbf{E}$ are all orthogonal to $\mathbf{t}$, so $\mathbf{Et}$ should be zero. The solution is the vector corresponding to the smallest singular value

of the SVD decomposition. Note the matrix $\mathbf{E}$ is not symmetric, so it will have both left and right eigenvectors

$$\mathbf{E} = \mathbf{U}\Lambda\mathbf{V}^T = \begin{bmatrix}\mathbf{u}_0 & \mathbf{u}_1 & \mathbf{u}_2\end{bmatrix} \begin{bmatrix}\sigma_0 & & \\ & \sigma_1 & \\ & & \sigma_2\end{bmatrix} \begin{bmatrix}\mathbf{v}_0^T \\ \mathbf{v}_1^T \\ \mathbf{v}_2^T\end{bmatrix} \tag{8.65}$$

where $\sigma_0 > \sigma_1 > \sigma_2$.

Set the unit vector along the translation: $\hat{\mathbf{t}} = \mathbf{u}_2$, the vector corresponding to the smallest left singular value.

- From wikipedia: "Not every arbitrary 3×3 matrix can be an essential matrix for some stereo cameras. To see this notice that it is defined as the matrix product of one rotation matrix and one skew-symmetric matrix, both 3×3. The skew-symmetric matrix must have two singular values which are equal and another which is zero. The multiplication of the rotation matrix does not change the singular values which means that also the essential matrix has two singular values which are equal and one which is zero. The properties described here are sometimes referred to as internal constraints of the essential matrix."

- NOTE: We can only recover the direction of the translation, not the magnitude. Impossible to recover distance between cameras, without a real world dimension. If we scale the world we will get the same pairs of images.

- Ideally, in a noise free world $\lambda_0 = \lambda_1 = 1$ and $\lambda_2 = 0$. We force these values to get a better estimate of the essential matrix.'

$$\mathbf{E} = \mathbf{U}\Lambda\mathbf{V}^T = \begin{bmatrix}\mathbf{u}_0 & \mathbf{u}_1 & \mathbf{u}_2\end{bmatrix} \begin{bmatrix}1 & & \\ & 1 & \\ & & 0\end{bmatrix} \begin{bmatrix}\mathbf{v}_0^T \\ \mathbf{v}_1^T \\ \mathbf{v}_2^T\end{bmatrix} \tag{8.66}$$

This is because see wikipedia quote above.

- Cross product can be written as matrix multiplication.

$$\hat{\mathbf{t}} \times \mathbf{r}_x = [\hat{\mathbf{t}}]_\times \mathbf{r}_0 = \begin{bmatrix}0 & -\hat{t}_z & \hat{t}_y \\ \hat{t}_z & 0 & -\hat{t}_x \\ -\hat{t}_y & \hat{t}_x & 0\end{bmatrix} \begin{bmatrix}r_{00} \\ r_{10} \\ r_{20}\end{bmatrix} \tag{8.67}$$

- Educated guess: This matrix can also be expressed using two other *orthogonal* unit vectors $\mathbf{s}_0$ and $\mathbf{s}_1$ chosen such that $\hat{\mathbf{t}} = \mathbf{s}_0 \times \mathbf{s}_1$.

$$[\hat{\mathbf{t}}]_\times = \begin{bmatrix} \mathbf{s}_0 & \mathbf{s}_1 & \hat{\mathbf{t}} \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & \\ & & 0 \end{bmatrix} \begin{bmatrix} 0 & -1 & \\ 1 & 0 & \\ & & 1 \end{bmatrix} \begin{bmatrix} \mathbf{s}_0^T \\ \mathbf{s}_1^T \\ \hat{\mathbf{t}}^T \end{bmatrix} \tag{8.68}$$

So the essential matrix can also be expressed as

$$\mathbf{E} = \hat{\mathbf{t}} \times \mathbf{R} = \begin{bmatrix} \mathbf{s}_0 & \mathbf{s}_1 & \hat{\mathbf{t}} \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & \\ & & 0 \end{bmatrix} \begin{bmatrix} 0 & -1 & \\ 1 & 0 & \\ & & 1 \end{bmatrix} \begin{bmatrix} \mathbf{s}_0^T \\ \mathbf{s}_1^T \\ \hat{\mathbf{t}}^T \end{bmatrix} \begin{bmatrix} & \mathbf{R} & \\ & & \end{bmatrix} \tag{8.69}$$

- Compare this with the SVD of the $\mathbf{E} = \mathbf{U}\Lambda\mathbf{V}^T$ and choose $\mathbf{s}_0 = \mathbf{u}_0$ and $\mathbf{s}_1 = \mathbf{u}_1$. Then we can see the correspondence between the rotation matrix and rest of the expression.

$$\begin{aligned} \mathbf{R}_{90^0}\mathbf{U}^T\mathbf{R} &= \mathbf{V}^T & (8.70) \\ \mathbf{R} &= \mathbf{U}\mathbf{R}_{90^0}^T\mathbf{V}^T & (8.71) \end{aligned}$$

- Possible four solutions of rotation matrix: the $\mathbf{R}_{90^0}$ can also be a rotation the other way, $\mathbf{R}_{-90^0}$ and the signs of the eigenvectors can be flipped in the SVD without change the decomposition.

$$\mathbf{R} = \pm\mathbf{U}\mathbf{R}_{\pm 90^0}^T\mathbf{V}^T \tag{8.72}$$

where

$$\mathbf{R}_{90^0} = \begin{bmatrix} 0 & -1 & \\ 1 & 0 & \\ & & 1 \end{bmatrix} \tag{8.73}$$

and

$$\mathbf{R}_{-90^0} = \begin{bmatrix} 0 & 1 & \\ -1 & 0 & \\ & & 1 \end{bmatrix} \tag{8.74}$$

Keep the two estimates of $\mathbf{R}$ whose determinant $\det \mathbf{R} = 1$. Pair them with two possibilities of the translation vector $\pm\mathbf{t}$ to arrive at 4 possible combinations.

- Use these estimates of the camera transformation, to estimate the coordinates of the corresponding pixels pairs. Pick the camera transformation for which we get the most number of postive z-coordinates (a condition also known as *chirality*).
- https://en.wikipedia.org/wiki/Essential_matrix

8.4 Correspondences

- Give a set of N points $\{\mathbf{x}_0^i | i = 1, \cdots, N\}$ in one image and M points $\{\mathbf{x}_1^i | i = 1, \cdots, M\}$ in the other image, how do you establish correspondences? These points could be detected using one of the many points detection methods that are in the literature such as the Harris detector you did for your assignment and the SIFT feature that we will study in this lecture.

- The mapping is one-to-one, but not onto. We have to allow for NULL mapping. There will be points in one image, not seen in the other image.

- Narrow baseline vs. wide-baseline (see Fig 4.2 and the paper that you read). There is lot of change in image structure both locally and globally for wide-baseline. For example, consider the local image in the corners of the roof. Local image characteristics differ a lot; the angles at which the edges come into the corner are different, i.e. the orientation of the edges will be different too. In general very hard to match wide-baseline images.

- There are two pieces of information that we can exploit to match the points (i) local description of the image and (ii) structure or relationships among the points.

- Possible local descriptors of points (Section 4.1.2 page 222)

    1. Just image intensity in a window, W, around the points and use the weighted sum of squared differences as the matching metric (Equation 4.1).

$$E_{WSSD} = \sum_{\mathbf{u} \in W} w(\mathbf{u})((I_0(\mathbf{x}_0^i + u) - I_1(\mathbf{x}_i^i + u))^2 \qquad (8.75)$$

       where $w(\mathbf{u})$ is the weight mask, typically Gaussian shaped.

    2. Use the feature vectors from the k-th level of a deep learning network, where k is chosen so that the footprint on the input image is equal to the mask.

    3. In most cases, however, the local appearance of features will change in orientation and scale, and sometimes even undergo

affine deformations. Extracting a local scale, orientation, or affine frame estimate and then using this to resample the patch before forming

4. Scale Invariant Feature Transform (SIFT) (page 217) of text book. **Use ppt about SIFT and HOG.**
   a) http://www.vlfeat.org/api/sift.html
   b) **Keypoint localization:** See picture in the book on page 218 (Fig 4.11). Detect extremas in scale space using Gaussian scale space. Use Difference of Gaussian (similar to Laplacian of a Gaussian) at different scales to detect "spots". The shape of this filter is an inverted Mexican hat.
   c) **Orientation Assignment:** (See Fig 4.12) A SIFT keypoint is a circular image region with an orientation. It is described by a geometric frame of four parameters: the keypoint center coordinates, its scale (the radius of the region), and its orientation.
   d) **Keypoint description** (See Fig 4.18) A 16x16 neighbourhood around the keypoint is taken. It is devided into 16 sub-blocks of 4x4 size. For each sub-block, 8 bin orientation histogram is created. So a total of 128 bin values are available. The 3-D histogram (consisting of $8\times4\times4=128$ bins) is stacked as a single 128-dimensional vector.
   e) The SIFT detector is invariant to translation, rotations, and re scaling of the image.
5. Other affine invariant detectors: http://www.robots.ox.ac.uk/~vgg/research/affine/.
6. Relational Distributions: Pairwise statisitics – rotation invariant.

- **Feature Matching Distances**: (See Figure 4.24) Euclidean distance (in feature space) with fixed threshold; Nearest Neighbor; Nearest neighbor with distance ratio matching;

- **Efficient Data Structure:**
  1. $\mathcal{O}(N^2)$ brute force matching is impractical for most applications.
  2. Use indexing structure, such as multi-dimensional hash table

3. (see Fig 4.27) k-d trees - see book by Hanen Samet. Binary decision tree - indexed by dimensions. search is $\mathcal{O}(\log N)$ instead of $O(N)$.

- **Feature Matching using RANSAC:**

Chapter 9

Optic Flow: Dense Motion

- **Instantaneous Motion:** What is the 2D projection of a 3D motion vector under perspective transform? Go answer this question we have to consider the *instantaneous* velocity models instead of the motion models, based on rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$ that we had considered earlier.

  1. The *instantaneous* velocity component related to translation is easily specified by $\mathbf{v} = \frac{d}{dt}\mathbf{t} = \dot{\mathbf{t}} = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^T$.
  2. The *instantaneous* velocity component related to rotation is captured by $\omega \times \mathbf{p}$, where $\mathbf{p} = \begin{bmatrix} X & Y & Z \end{bmatrix}$, the 3D coordinates. The angular velocity is captured by $\omega = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T$.
  3. The relationship between instantaneous rotation and the usual rotation matrix can be derived as follows.

$$\begin{aligned} \frac{\partial \mathbf{p}}{\partial t} &= \omega \times \mathbf{p} && (9.1) \\ &= \hat{\omega}\mathbf{p} && (9.2) \\ \mathbf{p}(t) &= \mathbf{p}(0)\exp(\hat{\omega}t) && (9.3) \end{aligned}$$

  So, we have

$$\begin{aligned} \mathbf{R} &= \exp(\hat{\omega}t) && (9.4) \\ &= I + \hat{\omega}t + \frac{1}{2!}\hat{\omega}^2 t^2 + \cdots && (9.5) \\ &= I + \hat{\omega}\sin(t) + \hat{\omega}^2(1 - \cos(t)) && (9.6) \end{aligned}$$

4. The instantaneous 3D velocity of the point is given by $\mathbf{v}+(\omega\times\mathbf{p})$, All coordinates are with respect to the camera coordinates.

- **Optic Flow** is the estimate of motion of pixels in the image based on intensity. It might or might not match the 2D motion vectors.

  1. Let $I(x, y, t)$ represent the image sequence. Note we need image frames that are finely sampled in time, at least at a faster than the rate of motion one is trying to capture. Movement of objects in the image frames should be in the order of a couple of pixels from frame to frame.
  2. We assume that the intensity of a point on an object does not change from one instant to the next. That is the total derivative of intensity with respect to time is zero.

     $$\frac{d}{dt}I(x, y, t) = 0 \tag{9.7}$$

     Note this is strictly not true. Intensity of a 3D point is dependent on the angle between the surface normal at the point and the illumination direction. Surface normal directions change with rotation. However, we will ignore this aspect in this first cut analysis.
  3.