6.801/6.866: Machine Vision, Lecture 14

Professor Berthold Horn, Ryan Sander, Tadayuki Yoshitake
MIT Department of Electrical Engineering and Computer Science
Fall 2020

These lecture summaries are designed to be a review of the lecture. Though I do my best to include all main topics from the lecture, the lectures will have more elaborated explanations than these notes. Therefore, if you're looking for the most rigorous review and treatment of these topics, we encourage you to rewatch the lecture videos. With that said, we hope these summaries are beneficial for your learning. If you have any feedback for these lecture summaries, please submit it **here**.

1 Lecture 14: Inspection in PatQuick, Hough Transform, Homography, Position Determination, Multi-Scale

In this lecture, we will continue our discussion of "PatQuick", the patent we discussed last time for object detection and pose estimation. We will focus on elements of this patent such as scoring functions and generalized degrees of freedom (DOF), and will use this as a segway into general linear transformations and homography. We will conclude our discussion with subsampling and Hough Transforms, which, at a high level, we can think of as a mapping from image space to parameter space.

1.1 Review of "PatQuick"

To frame our analysis and introduction of other technical machine vision concepts, let us briefly revisit the high-level ideas of "PatQuick". There were three main "objects" in this model:

- **Training/template image**. This produces a model consisting of probe points.
- **Model**, containing probe points.
- **Probe points**, which encode evidence for where to make gradient comparisons, i.e. to determine how good matches between the template image and the runtime image under the current pose configuration.

Once we have the model from the training step, we can summarize the process for generating matches as:

1. Loop over/sample from configurations of the pose space (which is determined and parameterized by our degrees of freedom), and modify the runtime image according to the current pose configuration.
2. Using the probe points of the model, compare the gradient direction (or magnitude, depending on the scoring function) to the gradient direction (magnitude) of the runtime image under the current configuration, and score using one of the scoring functions below.
3. Running this for all/all sampled pose configurations from the pose space produces a multidimensional scoring surface. We can find matches by looking for peak values in this surface.

A few more notes on this framework, before diving into the math:

- Training is beneficial here, because it allows for some degree of automated learning.
- Evidence collected from the probe points is cumulative and computed using many local operations.
- Accuracy is limited by the quantization level of the pose spanned. The non-redundant components of this pose space are:
  - 2D Translation, 2 DOF
  - Rotation, 1 DOF
  - Scaling, 1 DOF,
  - Skew, 1 DOF,

    - Aspect Ratio, 1 DOF

Together, the space of all these components compose a **general linear transformation**, or an **affine transformation**:

$$x' = a_{11}x + a_{12}y + a_{13}$$
$$y' = a_{21}x + a_{22}y + a_{23}$$

While having all of these options leads to a high degree of generality, it also leads to a huge number of pose configurations, even for coarse quantization. This is due to the fact that the number of configurations grows exponentially with the number of **DOF**.

1.1.1 Scoring Functions

Next, let us look at the scoring functions leveraged in this framework. Recall that there will also be random gradient matches in the background texture - we can compute this "probability" as "noise" given by N:

$$N = \frac{1}{2\pi}\int_0^{2\pi} R_{\text{dir}}(\theta)d\theta = \frac{3}{32}\ \textbf{(Using signed values)},\ \frac{6}{32}\ \textbf{(Using absolute values)}$$

Where the first value ($\frac{3}{32}$) corresponds to the probability of receiving a match if we randomly select a probe point's location, the second value corresponds to taking reverse polarity into account, and the function R_{dir} corresponds to the scoring function for the gradient direction, and takes as input the difference between two directions as two-dimensional vectors. Below are the scoring functions considered. Please take note of the following abbreviations:

- p_i denotes the two-dimensional location of the i^{th} probe point after being projected onto the runtime image.
- d_i denotes the direction of the probe point in the template image.
- w_i denotes the weight of the i^{th} probe point, which is typically set manually or via some procedure.
- $D(\cdot)$ is a function that takes a two-dimensional point as input, and outputs the direction of the gradient at this point.
- R_{dir} is a scoring function that takes as input the norm of the difference between two vectors representing gradient directions, and returns a scalar.
- $M(\cdot)$ is a function that computes the magnitude of the gradient in the runtime image at the point given by its two-dimensional argument.
- R_{mag} is a scoring function that takes as input the magnitude of a gradient and returns a scalar. In this case, R_{mag} saturates at a certain point, e.g. if $\lim_{x\to\infty} R(x) = K$ for some $K \in \mathbb{R}$, and for some $j \in \mathbb{R}$, $R(j) = K, R(j+\epsilon) = K \ \forall\ \epsilon \in [0,\infty)$.
- N corresponds to the "noise" term computed above from random matches.

With these terms specified, we are now ready to introduce the scoring functions:

1. **Piecewise-Linear Weighting with Direction and Normalization**:

   $$S_{1_a}(a) = \frac{\sum_i \max(w_i, 0) R_{\text{dir}}(||D(a+p_i) - d_i||_2)}{\sum_i \max(w_i, 0)}$$

   Quick note: the function $\max(0, w_i)$ is known as the Rectified Linear Unit (ReLU), and is written as $\text{ReLU}(w_i)$. This function comes up frequently in machine learning.

   - Works with "compiled probes". With these "compiled probes", we only vary translation - we have already mapped pose according to the other DOFs above.
   - Used in a "coarse step".

2. **Binary Weighting with Direction and Normalization**

   $$S_{1_a}(a) = \frac{\sum_i (w_i > 0) R_{\text{dir}}(||D(a+p_i) - d_i||_2)}{\sum_i (w_i > 0)}$$

   Where the predicate $(w_i > 0)$ returns 1 if this is true, and 0 otherwise.

3. **"Preferred Embodiment**:

$$S(a) = \frac{\sum_i (w_i > 0)(R_{\text{dir}}(||D(a+p_i) - d_i||_2) - N)}{(1-N)\sum_i (w_i > 0)}$$

4. **Raw Weights with Gradient Magnitude Scaling and No Normalization**

$$S_2(a) = \sum_i w_i M(a+p_i) R_{\text{dir}}(||D(a+p_i) - d_i||_2)$$

   Note that this scoring function is not normalized, and is used in the fine scanning step of the algorithm.

5. **Raw Weights with Gradient Magnitude Scaling and Normalization**

$$S_3(a) = \frac{\sum_i w_i M(a+p_i) R_{\text{dir}}(||D(a+p_i) - d_i||_2)}{\sum_i w_i}$$

1.1.2 Additional System Considerations

Let us focus on a few additional system features to improve our understanding of the system as well as other principles of machine vision:

- Why do some of these approaches use normalization, but not others? **Normalization is computationally-expensive**, and approaches that avoid a normalization step typically do this to speed up computation.
- For all these scoring functions, the granularity parameter is determined by decreasing the resolution until the system no longer performs well.
- We need to ensure we get the gradient direction right. So far, with just translation, this has not been something we need to worry about. But with our generalized linear transform space of poses, we may have to account for this. Specifically:
  - Translation
  - Rotation
  - Scaling

  do not affect the gradient directions. However:
  - Shear
  - Aspect ratio

  will both affect the gradient directions. We can account for this, however, using the following process:
  1. Compute the gradient in the runtime image prior to invoking any transformations on it.
  2. Compute the isophote in the pre-transformed runtime image by rotating 90 degrees from the gradient direction using the rotation matrix given by:

$$R_{\text{G}\rightarrow\text{I}} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

  3. Transform the isophote according to the generalized linear transformation above with the degrees of freedom we consider for our pose space.
  4. After computing this transformed isophote, we can find the transformed gradient by finding the direction orthogonal to the transformed isophote by rotating back 90 degrees using the rotation matrix given by:

$$R_{\text{I}\rightarrow\text{G}} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

1.1.3 Another Application of "PatQuick": Machine Inspection

Let us consider some elements of this framework that make it amenable and applicable for industrial machine part inspection:

- How do we distinguish between multiple objects (a task more generally known as multiclass object detection and classification)? We can achieve this by using **multiple models/template images, i.e. one model/template for each type of object we want to detect and find the relative pose of**.
- With this framework, we can also compute fractional matches - i.e. how well does one template match another object in the runtime image.
- We can also take an edge-based similarity perspective - we can look at the runtime image's edge and compare to edge matches achieved with the model.

1.2 Intro to Homography and Relative Poses

We will now shift gears and turn toward a topic that will also be relevant in the coming lectures on 3D. Let's revisit our perspective projection equations when we have a camera-centric coordinate system:

$$\frac{x}{f} = \frac{X_c}{Y_c}, \frac{y}{f} = \frac{Y_c}{Y_c}$$

Thus far, we have only considered camera-centric coordinate systems - that is, when the coordinates are from the point of view of the camera. But what if we seek to image points that are in a coordinate system defined by some world coordinate system that differs from the camera? Then, we can express these camera coordinates as:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ Z_W \end{bmatrix} + \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix}$$

$$\text{cam coord.} = \text{world2cam\_rot} \times \text{world coord.} + \text{world2cam\_trans}$$

$$\text{Where}\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \text{ is a rotation matrix in 3D.}$$

Therefore, in the general case, to find the perspective projection from world coordinates onto our image, we can combine the two previous equations, carrying out the matrix multiplication along the way:

$$\frac{x}{f} = \frac{X_c}{Z_c} = \frac{r_{11}X_W + r_{12}Y_W + r_{13}Z_W + X_0}{r_{31}X_W + r_{32}Y_W + r_{33}Z_W + Z_0}$$

$$\frac{y}{f} = \frac{Y_c}{Z_c} = \frac{r_{21}X_W + r_{22}Y_W + r_{23}Z_W + Y_0}{r_{31}X_W + r_{32}Y_W + r_{33}Z_W + Z_0}$$

Is there a way we can combine **rotation** and **translation** into a single operation? Let us consider a simple case in which the the points in our world coordinate system are coplanar in the three-dimensional plane $Z_W = 0$. Since the third column of the rotation corresponds to all zeros, we can rewrite our equation from the world coordinate frame to the camera frame as:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & X_0 \\ r_{21} & r_{22} & Y_0 \\ r_{31} & r_{32} & Z_0 \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ 1 \end{bmatrix} = \mathbf{T} \begin{bmatrix} X_W \\ Y_W \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ 0 \end{bmatrix} + \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix}$$

The vector $\begin{bmatrix} X_W & Y_W & 1 \end{bmatrix}^T$ expresses our (now 2D) world coordinate system in **homogeneous coordinates**, which have a 1 entry appended to the final element.

In this case, we can fold translation and rotation into a single matrix! We call this matrix **T**, and it is called a **Homography Matrix** that encodes both rotation and translation. We will revisit this concept when we begin our discussion of 3D transformations. Note that while our rotation matrix **R** is orthogonal, this homography matrix T is not necessarily.

1.2.1 How many degrees of freedom

For determining the relative pose between camera and world frames, let us consider the number of degrees of freedom:

- 3 for translation, since we can shift in x, y, and z
- 3 for rotation, since our rotations can preserve the xz axis, xy axis, and yz axis

If we have 9 entries in the rotation matrix and 3 in the translation vector (12 unknowns total), and only 6 degrees of freedom, then how do we solve for these entries? **There is redundancy - the rotation matrix has 6 constraints from orthonormality** (3 from constraining the rows to have unit size, and 3 from having each row being orthogonal to the other).

Mathematically, these constraints appear in our **Homography matrix T** as:

$$r_{11}^2 + r_{21}^2 + r_{31}^2 = 1 \textbf{ (Unit length constraint)}$$
$$r_{12}^2 + r_{22}^2 + r_{23}^2 = 1 \textbf{ (Unit length constraint)}$$
$$r_{11}r_{12} + r_{21}r_{22} + r_{31}r_{32} = 0 \textbf{ (Orthogonal columns)}$$

A few notes here about solving for our coefficients in $\mathbf{T}$:

- Do we need to enforce these constraints? Another option is to run least squares on the calibration data.
- We must be cognizant of the following: We only know the Homography matrix $\mathbf{T}$ up to a constant scale factor, since we are only interested in the ratio of the components of the camera coordinates for perspective projection.

1.3 Hough Transforms

Let us switch gears and talk about another way to achieve edge finding, but more generally the estimation of parameters for any parameterized surface.

Motivation: Edge and line detection for industrial machine vision. This was one of the first machine vision patents (submitted in 1960, approved in 1962). We are looking for lines in images, but our gradient-based methods may not necessarily work, e.g. due to non-contiguous lines that have "bubbles" or other discontinuities. These discontinuities can show up especially for smaller resolution levels.

Idea: The main idea of the **Hough Transform** is to intelligently map from image/surface space to parameter space for that surface. Let us walk through the mechanics of how parameter estimation works for some geometric objects.

Some notes on Hough Transforms:

- Hough transforms are often used as a subroutine in many other machine vision algorithms.
- Hough transforms actually generalize beyond edge and line detection, and extend more generally into any domain in which we map a parameterized surface in image space into parameter space in order to estimate parameters.

1.3.1 Hough Transforms with Lines

A line/edge in image space can be expressed (in two-dimensions for now, just for building intuition, but this framework is amenable for broader generalizations into higher-dimensional lines/planes): $y = mx + c$. Note that because $y = mx + c, m = \frac{y-c}{x}$ and $c = y - mx$. Therefore, this necessitates that:

- A line in image space maps to a singular point in Hough parameter space.
- A singular point in line space corresponds to a line in Hough parameter space.

To estimate the parameters of a line/accomplish edge detection, we utilize the following high-level procedure:

1. Map the points in the image to lines in Hough parameter space and compute intersections of lines.
2. Accumulate points and treat them as "evidence" using accumulator arrays.
3. Take peaks of these intersections and determine what lines they correspond to, since points in Hough parameter space define parameterizations of lines in image space. See the example below:

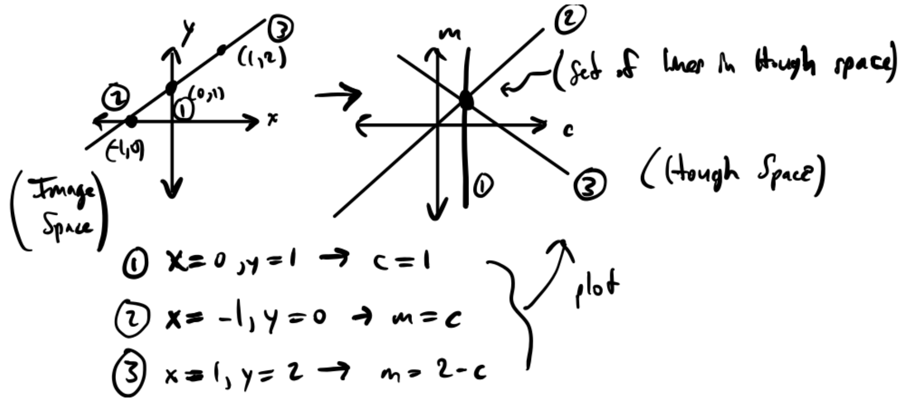

Figure 1: Example of finding parameters in Hough Space via the Hough Transform.

1.3.2 Hough Transforms with Circles

Let us look at how we can find parameters for circles with Hough transforms.

Motivating example: Localization with Long Term Evolution (LTE) Network. Some context to motivate this application further:

- LTE uses Time Division Multiplexing to send signals, a.k.a "everyone gets a slot".
- CDMA network does not use this.
- You can triangulate/localize your location based off of how long it takes to send signals to surrounding cellular towers.

We can see from the diagram below that we map our circles into Hough parameter space to compute the estimate of parameters. As we have seen in other problems we have studied in this class, we need to take more than one measurement. We cannot solve

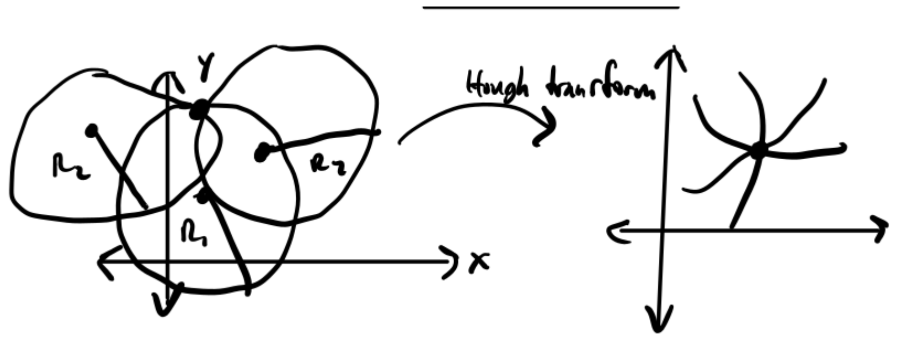

Figure 2: Example of using Hough Transforms to find the parameters of circles for LTE.

these problems with just one measurement, but a single measurement constrains the solution. Note that this problem assumes the radius is known.

1.3.3 Hough Transforms with Searching for Center Position and Radius

Another problem of interest is finding the center of position of a circle's radius R and its center position (x, y), which comprise the 3 dimensions in Hough parameter space. In Hough Transform space, this forms a cone that expands upward from $R_0 = 0$, where each cross-section of Z is the equation $(x^2 + y^2 = R^2)$ for the given values of x, y, and R.

Every time we find a point on the circle, we update the corresponding set of points on the cone that satisfy this equation.

The above results in many cone intersections with one another - as before, we collect evidence from these intersections, build a score surface, and compute the peak of this surface for our parameter estimates.

1.4 Sampling/Subsampling/Multiscale

Sampling is another important aspect for machine vision tasks, particularly for problems involving multiple scales, such as edge and line detection. **Sampling** is equivalent to working at **different scales**.

Why work at multiple scales?

- More efficient computation when resolution is lower, and is desirable if performance does not worsen.
- Features can be more or less salient at different resolutions, i.e. recall that edges are not as simple as step edges and often exhibit discontinuities or non-contiguous regions.

If we downsample our image by r_n along the rows and r_m along the columns, where $r_n, r_m \in (0, 1)$, then the total amount of work done (including the beginning image size) is given by the infinite geometric series:

$$\sum_{i=0}^{\infty} (r_n r_m)^i = \frac{1}{1 - r_n r_m}$$

$$\text{(Recall that } \sum_{i=0}^{\infty} r^i = \frac{1}{1-r} \text{ for } r \in (0,1))$$

What does the total work look like for some of these values?

- $r_n = r_m = r = \frac{1}{2}$

$$\text{work} = \frac{1}{1-r^2} = \frac{1}{1-\frac{1}{4}} = \frac{4}{3}$$

  But downsampling by $\frac{1}{2}$ each time is quite aggressive, and can lead to aliasing. Let us also look at a less aggressive sampling ratio.

- $r_n = r_m = r = \frac{1}{\sqrt{2}}$

$$\text{work} = \frac{1}{1-r^2} = \frac{1}{1-\frac{1}{2}} = 2$$

  How do we sample in this case? This is equivalent to taking every other sample in an image when we downsample. We can do this using a **checkerboard/chess board** pattern. We can even see the selected result as a square grid if we rotate our coordinate system by 45 degrees.

  The SIFT (Scale-Invariant Feature Transform) algorithm uses this less aggressive sampling technique. SIFT is a descriptor-based feature matching algorithm for object detection using a template image.