AA274A: Principles of Robot Autonomy I
Course Notes

Oct 24, 2019

12 Information extraction

Last time, we discussed how to extract information from images, specifically edges and corners. This lecture is focused on extracting information from the image. In particular, we will discuss how to identify geometric primitives within an image to assist in robot localization and mapping. Next, we will touch on a few parametric object recognition techniques[1].

12.1 Geometric feature extraction

Most of today's features extracted from ranging sensors are geometric primitives such as line segments or circles. The main reason for this is that for most other geometric primitives the parametric description of the features becomes too complex and no closed-form solution exists. In this section, we will focus on line extraction, since line segments are the simplest features to extract. As we have seen, lines are used to match laser scans for performing tasks like robot localization or automatic map building. There are three main problems in line extraction in unknown environments:

- How many lines are there?
- Which points belong to which line?
- Given the points that belong to a line, how to estimate the line model parameters?

For answering these questions, we will present the description of two popular line extraction algorithms for 2D range scans. But before describing the algorithms, we will first explain the line fitting problem, which answers the third question: "Given the points that belong to a line, how to estimate the line model parameters?"

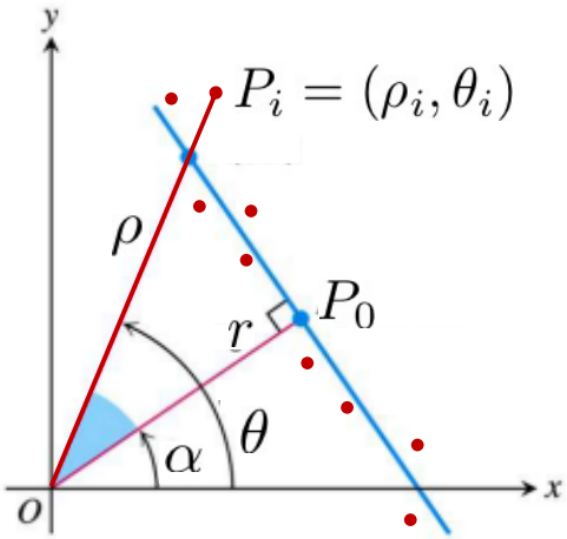


Figure 1: Representation of a line in polar coordinates

12.2 Line Fitting Problem

Geometric feature fitting is usually the process of comparing and matching measured sensor data against a predefined description, or template, of the expected feature. Usually, the system is overdetermined in that the number of sensor measurements exceeds the number of feature parameters to be estimated. Since the sensor measurements all have some error, there is no perfectly consistent solution and, instead, the problem is one of optimization. One can, for example, fit the feature that minimizes the discrepancy with all sensor measurements used (e.g,. *least-squares estimation*).

More formally, Our goal is to fit a line to a set of sensor measurements as shown in Figure 1. The Cartesian mapping of a point in the polar coordinates is given by

$$x = \rho \cos\theta, \quad y = \rho \sin\theta. \tag{1}$$

Let $P = (\rho, \theta)$ be an arbitrary point on the line. Since P, P_0, O determine a right triangle, we have

$$\rho \cos(\theta - \alpha) = r \tag{2}$$

or

$$x \cos\alpha + y \sin\alpha = r \tag{3}$$

where (r, α) are the parameters of the line. So our objective is to find a set of (r, α) that represent lines based on n ranging measurement points in polar coordinates $x_i = (\rho_i, \theta_i)$ produced by the robot's sensors.

[1]Unless stated otherwise, most of this lecture note is a direct excerpt from [SNS11])

How do we fit a line to measurement data? If there were no error, we would want to find a line for which all measurements lie on that line:

$$\rho\cos\theta\cos\alpha + \rho\sin\theta\sin\alpha - r = \rho\cos(\theta-\alpha) - r = 0. \tag{4}$$

But of course, there is measurement error, and so this quantity will not be zero, so there is no single line that passes through the set. Instead, we wish to select the best possible match, given some optimization criterion. Therefore, when the quantity is nonzero, this is a measure of the error between the measurement point (ρ, θ) and the line, specifically in terms of the minimum orthogonal distance between the point and the line. For each specific (ρ_i, θ_i), we can write the orthogonal distance d_i between (ρ_i, θ_i) and the line as

$$\rho_i\cos(\theta_i-\alpha) - r = d_i. \tag{5}$$

If we consider each measurement to be equally uncertain, we can sum the square of all errors together, for all measurement points, to quantify an overall fit between the line and all of the measurements:

$$S(r,\alpha) = \sum_{i=1}^{n} d_i^2 = \sum_{i=1}^{n}(\rho_i\cos(\theta_i-\alpha) - r)^2. \tag{6}$$

Our goal is to minimize S when selecting the line parameters (α, r). We can do so by solving the nonlinear equation system

$$\frac{\partial S}{\partial \alpha} = 0 \qquad \frac{\partial S}{\partial r} = 0 \tag{7}$$

This formalism is considered an unweighted least-squares solution because no distinction is made from among the measurements.

To make use of the variance σ_i that models the uncertainty regarding distance ρ_i of a particular sensor measurement, we compute an individual weight w_i for each measurement using the formula

$$w_i = \frac{1}{\sigma_i^2}. \tag{8}$$

Then, Equation 6 becomes

$$S = \sum w_i d_i^2 \sum w_i(\rho_i\cos(\theta_i-\alpha) - r)^2. \tag{9}$$

It can be shown that the solution to Equation 7 in this *weighted* least-squares sense is

$$r = \frac{1}{n}\sum_{i}^{n}\rho_i\cos(\theta_i-\alpha) \tag{10}$$

$$\alpha = \frac{1}{2}\arctan\left(\frac{\sum_i^n \rho_i^2\sin(2\theta_i) - \frac{2}{n}\sum_i^n\sum_j^n \rho_i\rho_j\cos\theta_i\sin\theta_j}{\sum_i^n \rho_i^2\cos(2\theta_i) - \frac{1}{n}\sum_i^n\sum_j^n \rho_i\rho_j\cos(\theta_i+\theta_j)}\right) + \frac{\pi}{2}. \tag{11}$$

In practice, Equation 11 uses the four-quadrant arctangent (atan2).

So far, we described how to fit a line feature given a set of range measurements. Unfortunately, the feature extraction process is significantly more complex than that. A mobile robot does indeed acquire a set of range measurements, but in general the range measurements are not all part of one line. Rather, only some of the range measurements should play a role in line extraction and, further, there may be more than one line feature represented in the measurement set. This leads to our next discussion on line extraction.

12.3 Line Extraction

As discussed earlier, we need to extract line data to fit and estimate lines in practical mobile robot application. The process of dividing up a set of measurements into subsets that can be interpreted one by one is termed *segmentation* and is the most important step of line extraction. We will look at three most popular line segmentation algorithms that have proven to work well in mobile robotics: Split-and-Merge, RANSAC, and Hough Transform.

12.3.1 Split-and-Merge

Split-and-Merge is the most popular line extraction algorithm. It is arguably the fastest, but not as robust to outliers as other algorithms. Algorithm 1 below gives a high level understanding of its working.

Algorithm 1: Split-and-Merge

Data: Set S consisting of all N points, a distance threshold $d > 0$
Result: L, a list of sets of points each resembling a line
$L \leftarrow (S)$, $i \leftarrow 1$;
while $i \leq len(L)$ **do**
  fit a line (r, α) to the set L_i;
  detect the point $P \in L_i$ with the maximum distance D to the line (r, α);
  if $D < d$ **then**
    $i \leftarrow i + 1$
  else
    split L_i at P into S_1 and S_2;
    $L_i \leftarrow S_1$; $L_{i+1} \leftarrow S_2$;
  end
end
Merge collinear sets in L;

There is another popular variant of the Split-and-Merge algorithm, the Iterative-End-Point-Fit variant, where the primary line is constructed by simply connecting the first and the last points and then the maximum distance of the data points are compared against a distance threshold.

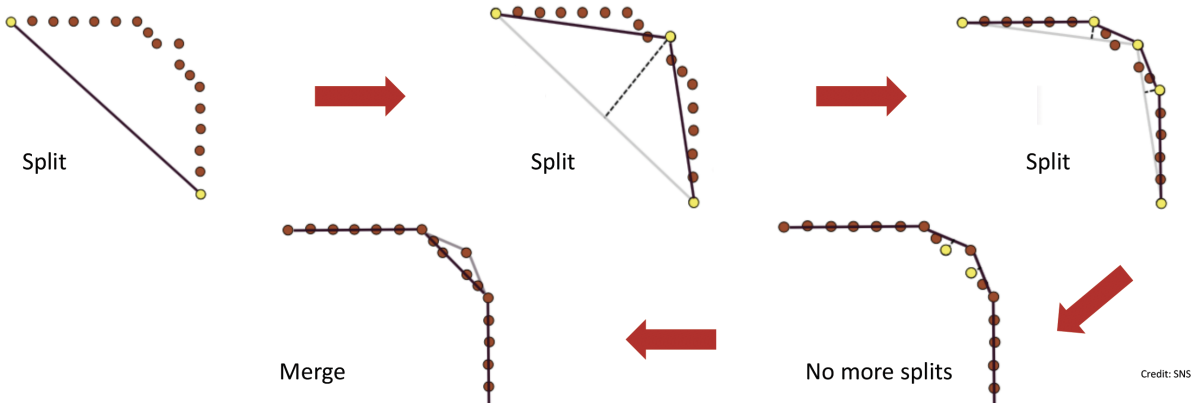


Figure 2: Iterative-end-point-fit variant of split-and-merge algorithm

12.3.2 Random Sample Consensus (RANSAC)

Random Sample Consensus (RANSAC) is an algorithm to estimate robustly the parameters of a model from a given data in the presence of outliers. Outliers are data that do not fit the model. Such outliers can be due to high noise in the data, wrong measurements, or they can more simply be points which come from other objects for which our mathematical model does not apply. For example, a typical laser scan in indoor environments may contain distinct lines from the surrounding walls but also points from other static and dynamic objects (like chairs or humans). In this case, an outlier is any entity which does not belong to a line (i.e., the chair, human, and so on).

RANSAC is an iterative method and is nondeterministic in that the probability to find a line free of outliers increases as more iterations are used. RANSAC is not restricted to line extraction from laser data but it can be generally applied to any problem where the goal is to identify the inliers which satisfy a predefined mathematical model. Typical applications in robotics are: line extraction from 2D range data (sonar or laser); plane extraction from 3D laser point clouds; and structure-from-motion, where the goal is to identify the image correspondences which satisfy a rigid body transformation.

Let us see how RANSAC works for the simple case of line extraction from 2D laser scan points. The algorithm starts by randmly selecting a sample of two points from the dataset. Then a line is constructed from these two points and the distance of all other points to this line is computed. The inliers set comprises all the points whose distance to the line is within a predefined threshold d. The algorithm then stores the inliers set and starts again by selecting another minimal set of two points at random. The procedure is iterated until a set with a maximum number of inliers is found, which is chosen as a solution to the problem. Figure 3 illustrates its working principle.

Because we cannot know in advance if the observed set contains the maximum number of inliers, the ideal would be to check all possible combinations of 2 points in a dataset of N points. The number of combinations is given by $N \cdot (N-1)/2$, which makes it computationally infeasible if N is too large. For example, in a laser scan of 360 points we would need to check

all $360 \cdot 359/2 = 64620$ possibilities!

At this point, a question arises: Do we really need to check all possibilities, or can we stop RANSAC after k iterations? The answer is that indeed we do not need to check all combinations but just a subset of them if we have a rough estimate of the percentage of inliers in our dataset. This can be done by thinking in a probabilistic way.

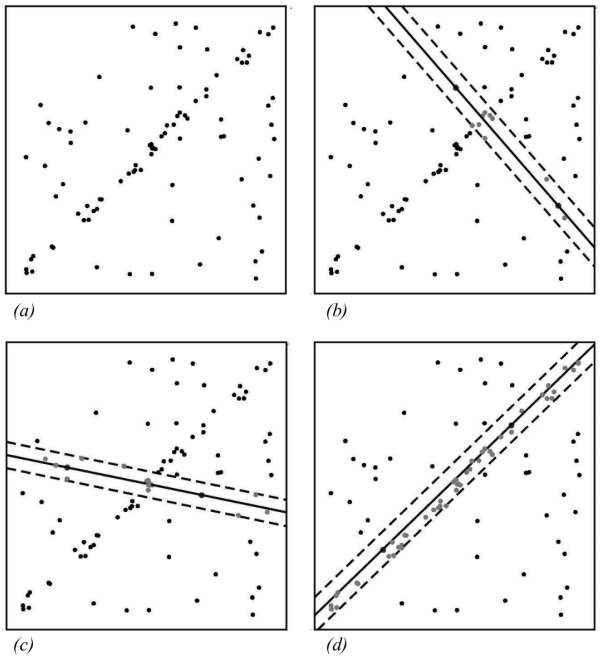


Figure 3: Working principle of RANSAC. (a) Dataset of N points. (b) Two points are randomly selected, a line is fitted through them, and the points within a predefined distance to it are identified. (c) The procedure is repeated (iterated) several times. (d) The set with the maximum number of inliers is chosen as a solution to the problem [SNS11])

Let p be the probability of finding a set of points free of outliers. Let w be the probability of selecting an inlier from our dataset of N points. Hence, w expresses the fraction of inliers in the data, that is, w = number of inliers/N. If we assume that the two points needed for estimating a line are selected independently, w is the probability that both points are inliers and 1–w is the probability that at least one of these two points is an outlier. Now, let k be the number of RANSAC iterations executed so far, then $(1 - w^2)^k$ will be the probability that RANSAC never selects two points that are both inliers. This probability must be equal to 1–p . Accordingly,

$$1 - p = (1 - w^2)^k, \tag{12}$$

and therefore

$$k = \frac{\log(1 - p)}{\log(1 - w^2)}. \tag{13}$$

This expression tells us that knowing the fraction w of inliers, after k RANSAC iterations we will have a probability p of finding a set of points free of outliers. For example, if we

want a probability of success equal to 99% and we know that the percentage of inliers in the dataset is 50%, then according to Equation 13 we could stop RANSAC after 16 iterations, which is much less than the number of all possible combinations that we had to check in the previous example! Also observe that in practice we do not need a precise knowledge of the fraction of inliers but just a rough estimate. More advanced implementations of RANSAC estimate the fraction of inliers by changing it adaptively iteration after iteration.

The main advantage of RANSAC is that it is a generic extraction method and can be used with many types of features once we have the feature model. It is also simple to implement. Another advantage is its ability to cope with large amount of outliers, even more than 50%. Clearly, if we want to extract multiple lines, we need to run RANSAC several time and remove sequentially all the lines extracted so far. A disadvantage of RANSAC is that when the maximum number of iterations k is reached, the solution obtained may not be the optimal one (i.e., the one with the maximum number of inliers). Furthermore, this solution may not even be the one that fits the data in the best way.

12.3.3 Hough transform

The next method is Hough transform. The key idea of this method is that each point votes for a set of plausible line parameters. A line has two parameters: (m, b). Given a point (x_i, y_i), the lines that could pass through this point are all (m, b) satisfying $y_i = mx_i + b$ or $b = -mx_i + y + i$. Thus, a point in image space (x, y) maps to a line in Hough space (m, b).

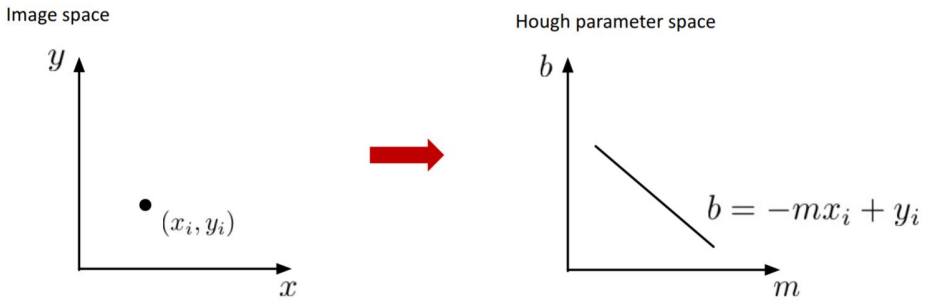


Figure 4

Two points on the same line in image space will yield two intersecting lines in Hough space.

This concept can be applied to point segmentation. If all the points lie exactly on the same line in image space, there there will be exactly one intersection for all the lines in Hough space. Of course, this will not be true given the realities of line fitting, so instead the "center" of the many intersections will have to be estimated. One technique is to discretize the Hough space. We can perform Hough transform using the following procedure:

1. Initialize an accumulator array $H(m, b)$ to zero
2. For each point (x_i, y_i), increment all cells that satisfy $b = -x_i m + y_i$

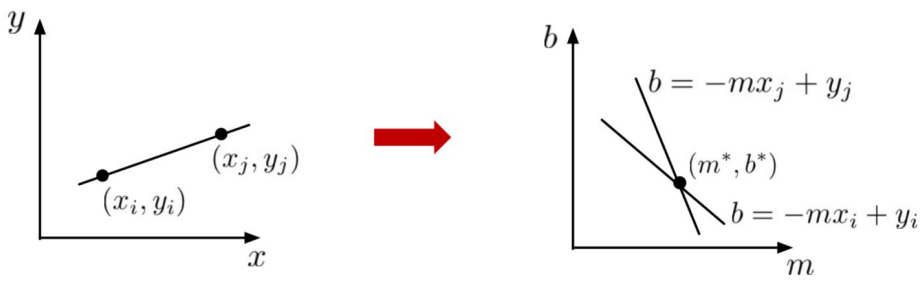


Figure 5

3. Local maxima in array $H(m, b)$ correspond to lines

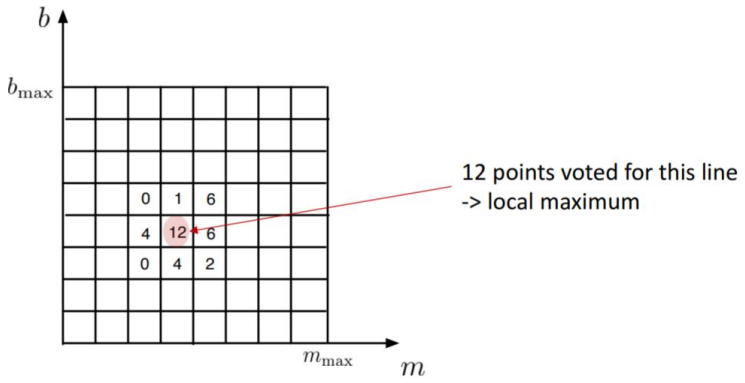
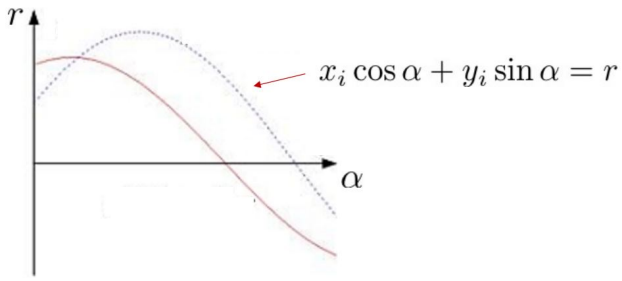
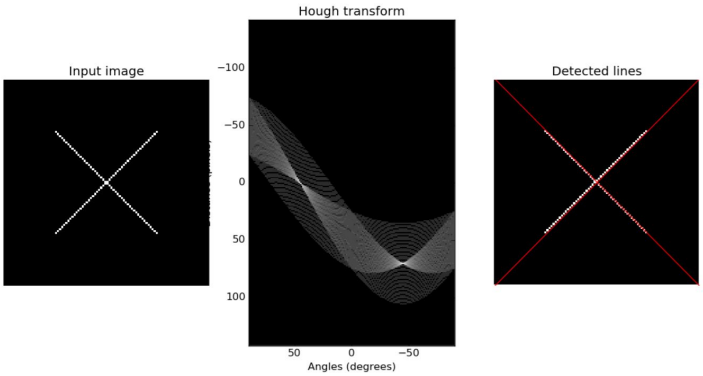


Figure 6

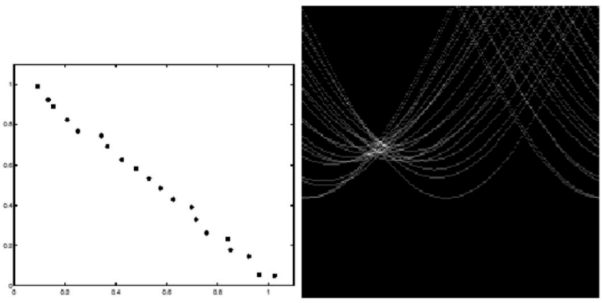
When performing discretization, there is an inherent tradeoff between range and resolution. This is problematic in the Hough space, since the slope m can range from $-\infty$ to ∞. The solution is to transform Cartesian coordinates to polar coordinates. The Hough space in polar coordinates now depends on α and r and avoids problems associated with infinity. Instead of a line in Cartesian Hough space, we are now presented with a sinusoidal curve in polar Hough space.



Below is an example featuring points that lie exactly on two lines. In Hough space two intersection points are clearly identified. These intersection points correspond with the parameters associated with the two red lines that fit the data exactly.



Given some noise, however, the intersection becomes less clear and some estimation is required.



12.3.4 Comparison of techniques

Three algorithms can be divided into two categories: deterministic and nondeterministic methods:

- Deterministic: Split-and-Merge, Hough transform
- Nondeterministic: RANSAC

RANSAC is nondeterministic because results can be different at every run, with random hypotheses generated each time.

Split-and-Merge can be intricate to implement. Its output depends heavily on parameter tuning and can take time to get right. However, it is the most computationally efficient method of the three. On the other hand, RANSAC is the most general of the three algorithms. It can extract lines, curves, circles, and other geometric features. If the data has a lot of noise, this is the algorithm of choice. However, it suffers from the fact that it is a probabilistic algorithm. Hough Transform is an elegant solution and can be seen as the middle ground of the above two techniques.

12.4 Object Recognition

12.4.1 Template Matching

Template Matching is a high-level machine vision technique that allows to identify the parts of an image (or multipleimages) that match the given image pattern[2]. It can be used in manufacturing as a part of quality control, a way to navigate a mobile robot, or as a way to detect edges in images. In this section we will see various methodologies used for implementing Template Matching.

Template Matching is a technique for finding areas of an image that match (are similar) to a template image (patch). We need two primary components:

- *Source image* (I): The image in which we expect to find a match to the template image.
- *Template image* (T): The patch image which will be compared to the template image. our goal is to detect best technique for the highest matching area.

Template matching is an area of profound interests in recent times. It has turned out to be a revolution in the field of computer vision, until the advent of deep learning. Template matching provides a new dimension into image-processing capabilities, although there have been many attempts to resolve different issues in this field. Template Matching is a high-level machine vision technique that allows one to identify the parts of an image (or multiple images) that match a given image pattern. Advanced template matching algorithms allow finding pattern occurrences regardless of their orientation and local brightness.

Naive Template Matching. Imagine that we are going to inspect a picture of a plug and our goal is to find its pins. We are provided with the pattern image representing the reference object we are looking for and the input image to be inspected. While there are more elegant ways to implement this, the most naive form of template matching is to position the pattern over the image at every possible location, and each time we will compute some numeric measure of similarity between the pattern and the image segment it currently overlaps. Finally we will identify the positions that yield the best similarity measures as the probable pattern positions.

Image Correlation Matching. One of the sub problems that occur in the specification above is calculating the similarity measure of the aligned pattern image and the overlapped segment of the input image, which is equivalent to calculating a similarity measure of two images of equal dimensions. This is a classical task, and a numeric measure of image similarity is called image *correlation*, which we discussed in lecture 11.

The fundamental method of calculating the image correlation is so called cross-correlation, which essentially is a simple sum of pairwise multiplications of corresponding pixel values of

[2]This section is a direct excerpt from [PKB13].

the images. Cross-correlation between two images I_1 and I_2 is written as

$$\text{Cross-Correlation}(I_1, I_2) = \sum_{i,j} I_1(i,j) \cdot I_2(i,j). \tag{14}$$

Though we may notice that the correlation value indeed seems to reflect the similarity of the images being compared, cross-correlation method is far from being robust. Its main drawback is that it is biased by changes in global brightness of the images - brightening of an image may sky-rocket its cross-correlation with another image, even if the second image is not at all similar.

Normalized Cross-Correlation. Normalized cross-correlation is an enhanced version of the classic cross-correlation method that introduces two improvements over the original one. For images I_1 and I_2, normalized correlation can be written as

$$\text{NCC} = \text{Correlation}(\hat{I}_1, \hat{I}_2) = \sum_{i,j} \hat{I}_1(i,j) \cdot \hat{I}_2(i,j) \tag{15}$$

where $\hat{I}_1 = \frac{I_1 - \bar{I}_1}{\sqrt{\sum (I_1 - \bar{I}_1)^2}}$ and $\hat{I}_2 = \frac{I_2 - \bar{I}_2}{\sqrt{\sum (I_2 - \bar{I}_2)^2}}$ and $\bar{I}_1, \bar{I}_2$ indicate the mean values of the entire image. Often, these results are invariant to the global brightness changes, i.e. consistent brightening or darkening of either image has no effect on the result (this is accomplished by subtracting the mean image brightness from each pixel value). The final correlation value is scaled to $[-1, 1]$ range, so that NCC of two identical images equals 1.0, while NCC of an image and its negation equals -1.0.

Normalized cross-correlation is widely used as an effective similarity measure for matching applications. Normalized cross-correlation is invariant to linear brightness and contrast variations, and its easy hardware implementation makes it useful for real-time applications. However, traditional correlation-based image matching methods will fail when there are large rotations or significant scale changes between the two images. This is because the normalized cross-correlation is sensitive to rotation and scale changes.

12.4.2 Image Pyramids

While template matching techniques could work well for feature detection of the same size, we often wish to change the resolution of an image before proceeding further[3]. Consider, for example, the task of finding a face in an image. Since we do not know the scale at which the face will appear, we need to generate a whole pyramid of differently sized images and scan each one for possible faces. Such a pyramid can also be very helpful in accelerating the search for an object by first finding a smaller instance of that object at a coarser level of the pyramid and then looking for the full resolution object only in the vicinity of coarse-level detections.

[3]This section is a direct excerpt from [Sze10].

Interpolation. In order to *interpolate* (or *upsample*) an image to a higher resolution, we need to select some interpolation kernel with which to convolve the image,

$$I'(x,y) = \sum_{i,j} F(i,j) \cdot I(x - ri, y - rj) \tag{16}$$

This formula is related to the discrete convolution formula from lecture 11, except that we replace i and j in I with ri and rj, where r is the upsampling rate.

Decimation. While interpolation can be used to increase the resolution of an image, decimation (downsampling) is required to reduce the resolution. To perform decimation, we first (conceptually) convolve the image with a low-pass filter (to avoid aliasing) and then keep every rth sample. In practice, we usually only evaluate the convolution at every rth sample,

$$I'(x,y) = \sum_{i,j} F(i,j) \cdot I(rx - i, ry - j). \tag{17}$$

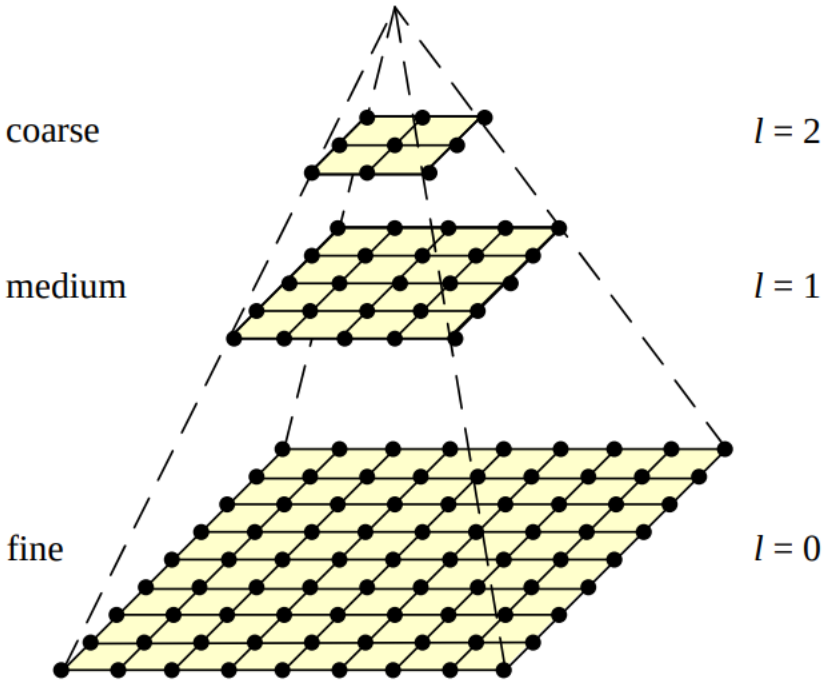


Figure 7: A traditional image pyramid: each level has half the resolution (width and height), and hence a quarter of the pixels, of its parent level [Sze10].

Multi-resolution representations. Now that we have described interpolation and decimation algorithms, we can build a complete image pyramid (Figure 7). As we mentioned before, pyramids can be used to accelerate coarse-to-fine search algorithms, to look for objects or patterns at different scales, and to perform multi-resolution blending operations.

12.5 Bag of Words

The idea behind "Bag of Words" is a way to simplify object representation as a collection of their subparts for purposes such as classification. The model originated in natural language processing, where we consider texts such as documents, paragraphs, and sentences as collections of words - effectively "bags" of words.

Consider a paragraph - a list of words and their frequencies can be considered a "bag of words" that represents the particular paragraph, which we can then use as a representation of the paragraph for tasks such as sentiment analysis, spam detection, and topic modeling. Although "Bag of Words" appears to be associated with language, the idea of simplifying complex objects into collections of their subparts can be extended to different types of objects. In Computer Vision, we can consider an image to be a collection of image features. By incorporating frequency counts of these features, we can apply the "Bag of Words" model towards images and use this for prediction tasks such as image classification and face detection.

There are two main steps for the "Bag of Words" method when applied to computer vision [Kri17].

1. Build a "dictionary" or "vocabulary" of features across many images - what kinds of common features exist in images? We can consider, for example, color scheme of the room, parts of faces suchas eyes, and different types of objects.

2. Given new images, represent them as histograms of the features we had collected - frequencies of the visual "words" in the vocabulary we have built.

12.5.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) were firstly introduced in the field of computer vision for image recognition in 1989 [LMB+90]. Since then, CNNs have boosted performance in image recognition and classification tasks.

A regular neural network consists of layers of neurons. Each neuron is a function $f : x \to y$. The simplest representation of this function is a linear model, where each function value is mapped with weight w and bias b. In other words, $f(x) = wx + b$. A more complex neural network may take the single neuron described above and "stack" them together such that one neuron passes its output as input into the next neuron, resulting in a more complex function [Ng].

While recent progress in computer vision with CNNs has significantly changed the future of robotics, a more thorough covering of CNNs will be given in AA 274B.

References

[Kri17] Ranjay Krishna. Computer vision:foundations and applications. `http://vision.stanford.edu/teaching/cs131_fall1718/files/cs131-class-notes.pdf`, 2017. (Accessed on 10/29/2019).

[LMB+90] Y. Le Cun, O. Matan, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jacket, and H. S. Baird. Handwritten zip code recognition with multilayer networks. In *[1990] Proceedings. 10th International Conference on Pattern Recognition*, volume ii, pages 35–40 vol.2, June 1990.

[Ng] Katanforoosh Kian Ng, Andrew. cs229-notes-deep_learning.pdf. `http://cs229.stanford.edu/notes/cs229-notes-deep_learning.pdf`. (Accessed on 10/29/2019).

[PKB13] Nazil Perveen, Darshan Kumar, and Ishan Bhardwaj. An overview on template matching methodologies and its applications. *International Journal of Research in Computer and Communication Technology*, 2(10):988–995, 2013.

[SNS11] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[Sze10] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.