ARTIFICIAL INTELLIGENCE

UNIT – IV

Syllabus

Natural Language for Communication: Phrase structure grammars, Syntactic Analysis, Augmented Grammars and semantic Interpretation, Machine Translation, Speech Recognition.

Perception: Image Formation, Early Image Processing Operations, Object Recognition by appearance, Reconstructing the 3D World, Object Recognition from Structural information, Using Vision.

Chapter – 1

Natural Language for Communication

INTRODUCTION

Communication is the intentional exchange of information brought about by the production SIGN and perception of signs drawn from a shared system of conventional signs. Most animals use signs to represent important messages: food here, predator nearby, approach, withdraw, let's mate.

4.1.1. PHRASE STRUCTURE GRAMMARS :

- The n-gram language models were based on sequences of words.
- The big issue for these models is **data sparsity**—with a vocabulary of, say, trigram probabilities to estimate, and so a corpus of even a trillion words will not be able to supply reliable estimates for all of them.
- We can address the problem of sparsity through generalization.
- Despite the exceptions, the notion of a **lexical category** (also known as a **part of speech**) such as *noun* or *adjective* is a useful generalization—useful in its own right, but more so when we string together lexical categories to form **syntactic categories** such as *noun phrase* or *verb phrase*, and combine these syntactic categories into trees representing the **phrase structure** of sentences: nested phrases, each marked with a category .

GENERATIVE CAPACITY :

- Grammatical formalisms can be classified by their **generative capacity**: the set of languages they can represent.
- Chomsky (1957) describes four classes of grammatical formalisms that differ only in the form of the rewrite rules.
- The classes can be arranged in a hierarchy, where each class can be used to describe all the languages that can be described by a less powerful class, as well as some additional languages.
- Here we list the hierarchy, most powerful class first:

1. **Recursively enumerable** grammars use unrestricted rules: both sides of the rewrite rules can have any number of terminal and nonterminal symbols, as in the rule A B C → D E.
    - These grammars are equivalent to Turing machines in their expressive power.
2. **Context-sensitive grammars** are restricted only in that the right-hand side must contain at least as many symbols as the left-hand side.
    - The name "contextsensitive" comes from the fact that a rule such as A X B → A Y B says that an X can be rewritten as a Y in the context of a preceding A and a following B.

    Context-sensitive grammars can represent languages such as (a sequence of n copies of a followed by the same number of bs and then cs).

3. In **context-free grammars** (or **CFG**s), the left-hand side consists of a single nonterminal symbol. Thus, each rule licenses rewriting the nonterminal as the right-hand side in *any* context.

   CFGs are popular for natural-language and programming-language grammars, although it is now widely accepted that at least some natural languages have constructions that are not context-free (Pullum, 1991).

   Context-free grammars can represent $a^n b^n$, but not $a^n b^n c^n$.

4. **Regular** grammars are the most restricted class. Every rule has a single nonterminal on the left-hand side and a terminal symbol optionally followed by a nonterminal on the right- hand side.

   Regular grammars are equivalent in power to finite state machines. They are poorly suited for programming languages, because they cannot represent constructs such as balanced opening and closing parentheses .

   The closest they can come is representing a▯b▯, a sequence of any number of as followed by any number of bs.

- There have been many competing language models based on the idea of phrase structure; we will describe a popular model called the **probabilistic context-free grammar**, or PCFG.
- A **grammar** is a collection of rules that defines a **language** as a set of allowable strings of words. Probabilistic means that the grammar assigns a probability to every string.
- Here is a PCFG rule:

  VP → Verb [0.70]

  VP NP [0.30]

  - Here VP (*verb phrase*) and NP (*noun phrase*) are **non-terminal symbols**. The grammar also refers to actual words, which are called **terminal symbols**.
  - This rule is saying that with probability 0.70 a verb phrase consists solely of a verb, and with probability 0.30 it is a VP followed by an NP.

(i) **The lexicon of**

- First we define the **lexicon**, or list of allowable words. The words are grouped into the lexical categories familiar to dictionary users: nouns, pronouns, and names to denote things; verbs to denote events; adjectives to modify nouns; adverbs to modify verbs; and function words: articles (such as *the*), prepositions (*in*), and conjunctions (*and*).
- Each of the categories ends in . . . to indicate that there are other words in the category.
- For nouns, names, verbs, adjectives, and adverbs, it is infeasible even in principle to list all the words. Not only are there tens of thousands of members in each class, but new ones–like *iPod* or

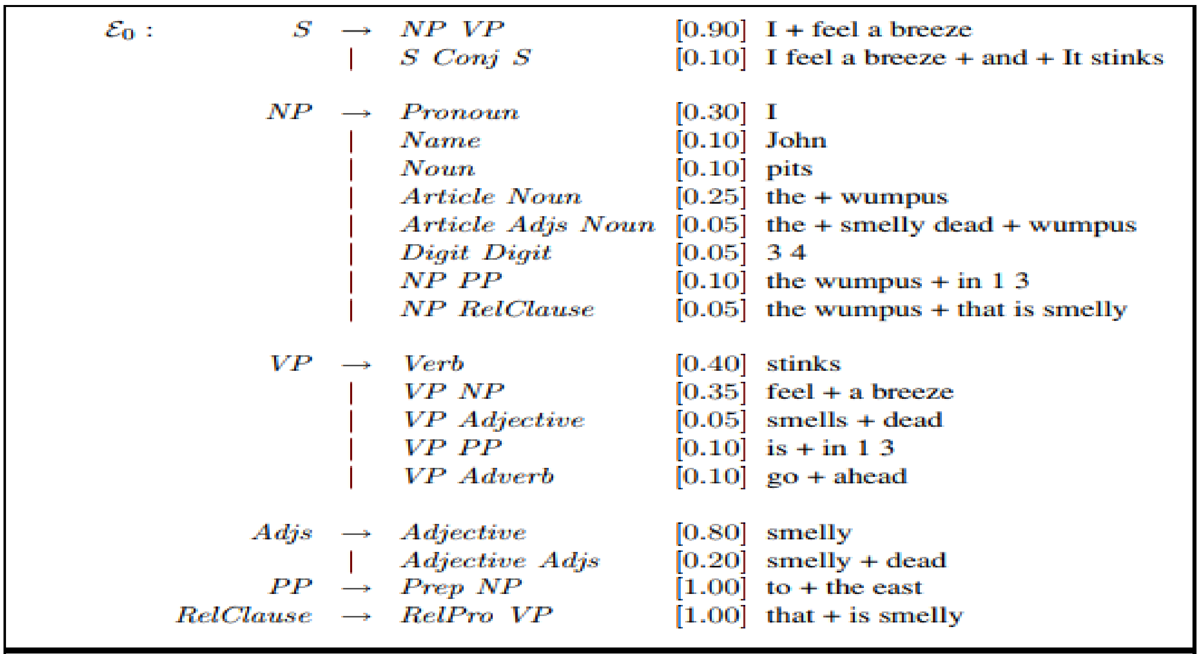
biodiesel—are being added constantly.

- These five categories are called **open classes**.
- For the categories of pronoun, relative pronoun, article, preposition, and conjunction we could have listed all the words with a little more work. These are called **closed classes**; they have a small number of words (a dozen or so).
- Closed classes change over the course of centuries, not months. For example, “thee” and “thou” were commonly used pronouns in the 17th century, were on the decline in the 19th, and are seen today only in poetry and some regional dialects.

(ii) **The Grammar of $\mathcal{E}_0$**

- The next step is to combine the words into phrases.

A grammar for $\mathcal{E}_0$ with rules for each of the six syntactic categories and an example for each rewrite rule.



$\mathcal{E}_0$:	S	$\rightarrow$	$NP\ VP$	[0.90]	I + feel a breeze
		$\mid$	$S\ Conj\ S$	[0.10]	I feel a breeze + and + It stinks
	NP	$\rightarrow$	$Pronoun$	[0.30]	I
		$\mid$	$Name$	[0.10]	John
		$\mid$	$Noun$	[0.10]	pits
		$\mid$	$Article\ Noun$	[0.25]	the + wumpus
		$\mid$	$Article\ Adjs\ Noun$	[0.05]	the + smelly dead + wumpus
		$\mid$	$Digit\ Digit$	[0.05]	3 4
		$\mid$	$NP\ PP$	[0.10]	the wumpus + in 1 3
		$\mid$	$NP\ RelClause$	[0.05]	the wumpus + that is smelly
	VP	$\rightarrow$	$Verb$	[0.40]	stinks
		$\mid$	$VP\ NP$	[0.35]	feel + a breeze
		$\mid$	$VP\ Adjective$	[0.05]	smells + dead
		$\mid$	$VP\ PP$	[0.10]	is + in 1 3
		$\mid$	$VP\ Adverb$	[0.10]	go + ahead
	$Adjs$	$\rightarrow$	$Adjective$	[0.80]	smelly
		$\mid$	$Adjective\ Adjs$	[0.20]	smelly + dead
	PP	$\rightarrow$	$Prep\ NP$	[1.00]	to + the east
	$RelClause$	$\rightarrow$	$RelPro\ VP$	[1.00]	that + is smelly

Figure 23.2 The grammar for $\mathcal{E}_0$, with example phrases for each rule. The syntactic categories are sentence (S), noun phrase (NP), verb phrase (VP), list of adjectives ($Adjs$), prepositional phrase (PP), and relative clause ($RelClause$).

4.1.2. SYNTACTIC ANALYSIS (PARSING) :

- **Parsing** is the process of analyzing a string of words to uncover its phrase structure, according to the rules of a grammar.
- Consider the following two sentences:

1. Have the students in section 2 of Computer Science 101 take the exam.

2. Have the students in section 2 of Computer Science 101 taken the exam?

- Even though they share the first 10 words, these sentences have very different parses, because the first is a command and the second is a question.
- By using left-to-right parsing algorithm would have to guess whether the first word is part of a command or a question and will not be able to tell if the guess is correct until at least the eleventh word, *take* or *taken.*
- If the algorithm guesses wrong, it will have to backtrack all the way to the first word and reanalyze the whole sentence under the other interpretation.
- To avoid this source of inefficiency we can use dynamic programming: *every time we analyze a substring, store the results so we won't have to reanalyze it later.*
- For example, once we discover that "the students in section 2 of Computer Science 101" is an NP, we can record that result in a data structure known as a **chart**.
- Algorithms that do this are called **chart parsers**.
- There are many types of chart parsers; we describe a bottom-up version called the **CYK algorithm**, after its inventors, **John Cocke, Daniel Younger, and Tadeo Kasami.**

CYK algorithm :

```
function CYK-PARSE(words, grammar) returns P, a table of probabilities
  N ← LENGTH(words)
  M ← the number of nonterminal symbols in grammar
  P ← an array of size [M, N, N], initially all 0
  /* Insert lexical rules for each word */
  for i = 1 to N do
    for each rule of form (X → words_i [p]) do
      P[X, i, 1] ← p
  /* Combine first and second parts of right-hand sides of rules, from short to long */
  for length = 2 to N do
    for start = 1 to N − length + 1 do
      for len1 = 1 to N − 1 do
        len2 ← length − len1
        for each rule of the form (X → Y Z [p]) do
          P[X, start, length] ← MAX(P[X, start, length],
                               P[Y, start, len1] × P[Z, start + len1, len2] × p)
  return P
```

Figure 23.5 The CYK algorithm for parsing. Given a sequence of words, it finds the most probable derivation for the whole sequence and for each subsequence. It returns the whole table, P, in which an entry $P[X, start, len]$ is the probability of the most probable X of length len starting at position $start$. If there is no X of that size at that location, the probability is 0.

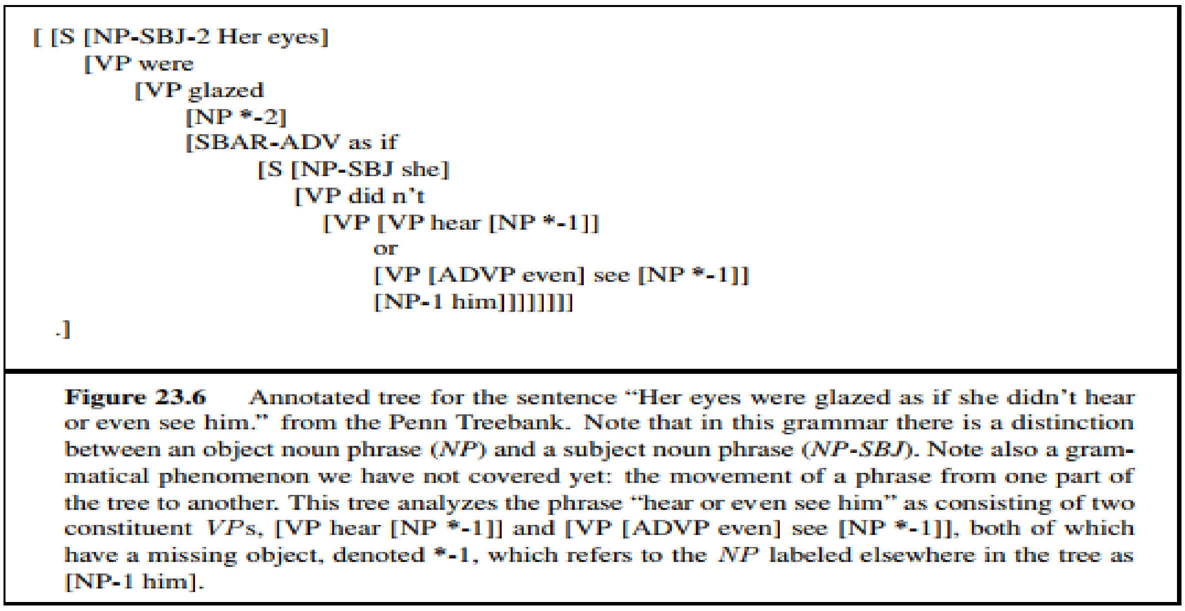
- The CYK algorithm requires a grammar with all rules in one of two very specific formats: lexical rules of the form X → **word**, and syntactic rules of the form X → Y Z .
- This grammar format, called **Chomsky Normal Form**, may seem restrictive, but it is not: any context-free grammar can be automatically transformed into Chomsky Normal Form.

(i) **Learning probabilities for PCFGs :**

- A PCFG has many rules, with a probability for each rule.
- This suggests that **learning** the grammar from data might be better than a knowledge engineering approach.
- Learning is easiest if we are given a corpus of correctly parsed sentences, commonly called a **treebank**.

The Penn Treebank is the best known; it consists of 3 million words which have been annotated with part of speech and parse-tree structure, using human labor assisted by some automated tools.

- Annotated tree from the Penn Treebank :



```
[ [S [NP-SBJ-2 Her eyes]
     [VP were
          [VP glazed
               [NP *-2]
               [SBAR-ADV as if
                    [S [NP-SBJ she]
                       [VP did n't
                          [VP [VP hear [NP *-1]]
                               or
                               [VP [ADVP even] see [NP *-1]]
                               [NP-1 him]]]]]]]]
  .]
```

Figure 23.6 Annotated tree for the sentence "Her eyes were glazed as if she didn't hear or even see him." from the Penn Treebank. Note that in this grammar there is a distinction between an object noun phrase (*NP*) and a subject noun phrase (*NP-SBJ*). Note also a grammatical phenomenon we have not covered yet: the movement of a phrase from one part of the tree to another. This tree analyzes the phrase "hear or even see him" as consisting of two constituent *VP*s, [VP hear [NP *-1]] and [VP [ADVP even] see [NP *-1]], both of which have a missing object, denoted *-1, which refers to the *NP* labeled elsewhere in the tree as [NP-1 him].

- Given a corpus of trees, we can create a PCFG just by counting (and smoothing).
- In the example above, there are two nodes of the form [S[NP . . .][VP . . .]]. We would count these, and all the other subtrees with root S in the corpus.

  If there are 100,000 S nodes of which 60,000 are of this form, then we create the rule:

  S → NP VP [0.60] .

(ii) Comparing context-free and Markov models :

- The problem with PCFGs is that they are context-free.
- That means that the difference between P (“eat a banana”) and P (“eat a bandanna”) depends only on P (Noun → “banana”) versus
  P (Noun → “bandanna”) and not on the relation between “eat” and the respective objects.
- A Markov model of order two or more, given a sufficiently large corpus, *will* know that “eat a banana” is more probable.
- We can combine a PCFG and Markov model to get the best of both. The simplest approach is to estimate the probability of a sentence with the geometric mean of the probabilities computed by both models.

  Another problem with PCFGs is that they tend to have too strong a preference for shorter sentences.

4.1.3. AUGMENTED GRAMMARS AND SEMANTIC INTERPRETATION :

- In this concept, we see how to extend context-free grammars.

(i) **Lexicalized PCFGs** :

To get at the relationship between the verb “eat” and the nouns “banana” versus “bandanna,” we can use a **lexicalized PCFG**, in which the probabilities for a rule depend on the relationship between words in the parse tree, not just on the adjacency of words in a sentence.

- Of course, we can’t have the probability depend on every word in the tree, because we won’t have enough training data to estimate all those probabilities.
- It is useful to introduce the notion of the **head** of a phrase—the most important word. Thus, “eat” is the head of the VP “eat a banana” and “banana” is the head of the NP “a banana.”
- We use the notation VP(v) to denote a phrase with category VP whose head word is v. We say that the category VP is **augmented** with the head variable v.

  Here is an **augmented grammar** that describes theverb–object relation:

$$\begin{array}{lcll}
VP(v) & \rightarrow & Verb(v)\ NP(n) & [P_1(v,n)] \\
VP(v) & \rightarrow & Verb(v) & [P_2(v)] \\
NP(n) & \rightarrow & Article(a)\ Adjs(j)\ Noun(n) & [P_3(n,a)] \\
Noun(\mathbf{banana}) & \rightarrow & \mathbf{banana} & [p_n] \\
\ldots & & & \ldots
\end{array}$$

(ii) **Formal definition of augmented grammar rules:**

- Augmented rules are complicated, so we will give them a formal definition by showing how an augmented rule can be translated into a logical sentence.

- The sentence will have the form of a definite clause, so the result is called a **definite clause grammar**, or DCG.

  That gives us

$$\begin{aligned} &Article(a, s_1) \land Adjs(j, s_2) \land Noun(n, s_3) \land Compatible(j, n) \\ &\qquad \Rightarrow NP(n, Append(s_1, s_2, s_3)) \,. \end{aligned}$$

- This definite clause says that if the predicate Article is true of a head word a and a string s1, and Adjs is similarly true of a head word j and a string s2, and Noun is true of a head word n and a string s3, and if j and n are compatible, then the predicate NP is true of the head
  word n and the result of appending strings s1, s2, and s3.
- The translation from grammar rule to definite clause allows us to talk about parsing as logical inference.
- This makes it possible to reason about languages and strings in many different ways.

  For example, it means we can do bottom-up parsing using forward chaining or top-down parsing using backward chaining .
- In fact, parsing natural language with DCGs was one of the first applications of (and motivations for) the Prolog logic programming language.
- It is sometimes possible to run the process backward and do **language generation** as well as parsing.

(iii) **Case agreement and subject–verb agreement:**

- We splitting NP into two categories, NPS and NPO, to stand for noun phrases in the subjective and objective case, respectively.
- We would also need to split the category Pronoun into the two categories PronounS (which includes "I") and PronounO (which includes "me").
- The top part of Figure shows the grammar for **case agreement**; we call the resulting language $\mathcal{E}_1$

$$
\begin{array}{lrcl}
\mathcal{E}_1: & S & \rightarrow & NP_S\ VP \mid \ \ldots \\
 & NP_S & \rightarrow & Pronoun_S \mid Name \mid Noun \mid \ \ldots \\
 & NP_O & \rightarrow & Pronoun_O \mid Name \mid Noun \mid \ \ldots \\
 & VP & \rightarrow & VP\ NP_O \mid \ \ldots \\
 & PP & \rightarrow & Prep\ NP_O \\
 & Pronoun_S & \rightarrow & \mathbf{I} \mid \mathbf{you} \mid \mathbf{he} \mid \mathbf{she} \mid \mathbf{it} \mid \ \ldots \\
 & Pronoun_O & \rightarrow & \mathbf{me} \mid \mathbf{you} \mid \mathbf{him} \mid \mathbf{her} \mid \mathbf{it} \mid \ \ldots \\
 & & & \ldots
\end{array}
$$

$$
\begin{array}{lrcl}
\mathcal{E}_2: & S(head) & \rightarrow & NP(Sbj, pn, h)\ VP(pn, head) \mid \ \ldots \\
 & NP(c, pn, head) & \rightarrow & Pronoun(c, pn, head) \mid \ Noun(c, pn, head) \mid \ \ldots \\
 & VP(pn, head) & \rightarrow & VP(pn, head)\ NP(Obj, p, h) \ \mid \ \ldots \\
 & PP(head) & \rightarrow & Prep(head)\ NP(Obj, pn, h) \\
 & Pronoun(Sbj, 1S, \mathbf{I}) & \rightarrow & \mathbf{I} \\
 & Pronoun(Sbj, 1P, \mathbf{we}) & \rightarrow & \mathbf{we} \\
 & Pronoun(Obj, 1S, \mathbf{me}) & \rightarrow & \mathbf{me} \\
 & Pronoun(Obj, 3P, \mathbf{them}) & \rightarrow & \mathbf{them} \\
 & & & \ldots
\end{array}
$$

Figure 23.7 Top: part of a grammar for the language $\mathcal{E}_1$, which handles subjective and objective cases in noun phrases and thus does not overgenerate quite as badly as $\mathcal{E}_0$. The portions that are identical to $\mathcal{E}_0$ have been omitted. Bottom: part of an augmented grammar for $\mathcal{E}_2$, with three augmentations: case agreement, subject–verb agreement, and head word. *Sbj*, *Obj*, *1S*, *1P* and *3P* are constants, and lowercase names are variables.

- Unfortunately, E1 still overgenerates. English requires **subject–verb agreement** for person and number of the subject and main verb of a sentence.

  For example, if "I" is the subject, then "I smell" is grammatical, but "I smells" is not. If "it" is the subject, we get the reverse.

(iv) Semantic interpretation :

To show how to add semantics to a grammar, we start with an example that is simpler than English: the semantics of arithmetic expressions.

$$
\begin{array}{l}
Exp(x) \ \rightarrow \ Exp(x_1)\ Operator(op)\ Exp(x_2)\ \{x = Apply(op, x_1, x_2)\} \\
Exp(x) \ \rightarrow \ (\ Exp(x)\ ) \\
Exp(x) \ \rightarrow \ Number(x) \\
Number(x) \ \rightarrow \ Digit(x) \\
Number(x) \ \rightarrow \ Number(x_1)\ Digit(x_2)\ \{x = 10 \times x_1 + x_2\} \\
Digit(x) \ \rightarrow \ x\ \{0 \leq x \leq 9\} \\
Operator(x) \ \rightarrow \ x\ \{x \in \{+, -, \div, \times\}\}
\end{array}
$$

Figure 23.8 A grammar for arithmetic expressions, augmented with semantics. Each variable x_i represents the semantics of a constituent. Note the use of the $\{test\}$ notation to define logical predicates that must be satisfied, but that are not constituents.

- Figure shows a grammar for arithmetic expressions, where each rule is augmented with a variable indicating the semantic interpretation of the phrase.
- The semantics of a digit such as "3" is the digit itself. The semantics of an expression such as "3 + 4" is the operator "+" applied to the semantics of the phrase "3" and the phrase "4."

The rules obey the principle of **compositional semantics.**

- The semantics of a phrase is a function of the semantics of the subphrases. Figure shows the parse tree for 3 + (4 ÷ 2) according to this grammar. The root of the parse tree is Exp(5), an expression whose semantic interpretation is 5.

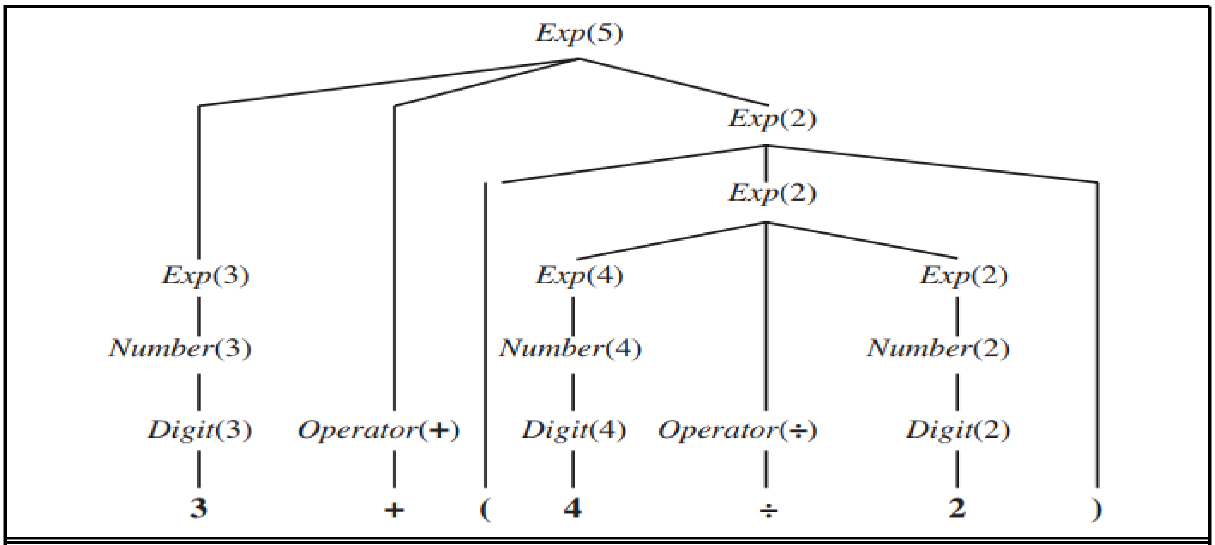


Figure 23.9 Parse tree with semantic interpretations for the string "3 + (4 ÷ 2)".

(v) Complications :

- The grammar of real English is endlessly complex. We will briefly mention some examples.

1. **Time and tense:**

Suppose we want to represent the difference between "John loves Mary" and "John loved Mary."

- English uses verb tenses (past, present, and future) to indicate the relative time of an event. One good choice to represent the time of events is the event calculus notation
- In event calculus we have

John loves mary: E1 ⍰ Loves(John, Mary) ⍰ During(Now, Extent(E1))
John loved mary: E2 ⍰ Loves(John, Mary) ⍰ After(Now, Extent(E2)) .

2. **Quantification** :

- Consider the sentence "Every agent feels a breeze."

The sentence has only one syntactic parse under E0, but it is actually semantically ambiguous; the preferred meaning is "For every agent there exists a breeze that the agent feels," but an acceptable alternative meaning is "There exists a breeze that every agent feels.

3. Pragmatics:

- We have shown how an agent can perceive a string of words and use a grammar to derive a set of possible semantic interpretations.

4. Long-distance dependencies:

Questions introduce a new grammatical complexity. In “Who did the agent tell you to give the gold to?” the final word “to” should be parsed as [PP to], where the “ ” denotes a gap or **trace** where an NP is missing; the missing NP is licensed by the first word of the sentence, “who.”

5. Ambiguity :

In some cases, hearers are consciously aware of ambiguity in an utterance.

Types of ambiguities :

→ **Lexical ambiguity**, in which a word has more than one meaning.

→ **Syntactic ambiguity,** refers to a phrase that has multiple parses.

→**Semantic ambiguity,** The syntactic ambiguity leads to a **semantic ambiguity**, because one parse means the other.

- **Disambiguation,** is the process of recovering the most probable intended meaning of an utterance.
- To do disambiguation properly, we need to combine four models:

→ **world model**

→ **mental model**

→ **language model**

→ **acoustic mode**

4.1.4. MACHINE TRANSLATION :

- Machine translation is the automatic translation of text from one natural language (the source) to another (the target).

It was one of the first application areas envisioned for computers (Weaver, 1949), but it is only in the past decade that the technology has seen widespread usage.

- Historically, there have been **three main applications** of machine translation.

→ ***Rough translation***, as provided by free online services, gives the “gist” of a foreign sentence or document, but contains errors.

→ ***Pre-edited translation*** is used by companies to publish their documentation and sales materials in multiple languages.

The original source text is written in a constrained language that is easier to translate automatically, and the results are usually edited by a human to correct any errors.

→ ***Restricted-source translation*** works fully automatically, but only on highly stereotypical language, such as a weather report.

▪ Translation is difficult because, in the fully general case, it requires in-depth understanding of the text. This is true even for very simple texts—even "texts" of one word.

(i) **Machine translation systems** :

▪ All translation systems must model the source and target languages, but systems vary in the type of models they use.

▪ Some systems attempt to analyze the source language text all the way into an interlingua knowledge representation and then generate sentences in the target language from that representation.

▪ This is difficult because it involves **three unsolved problems**:

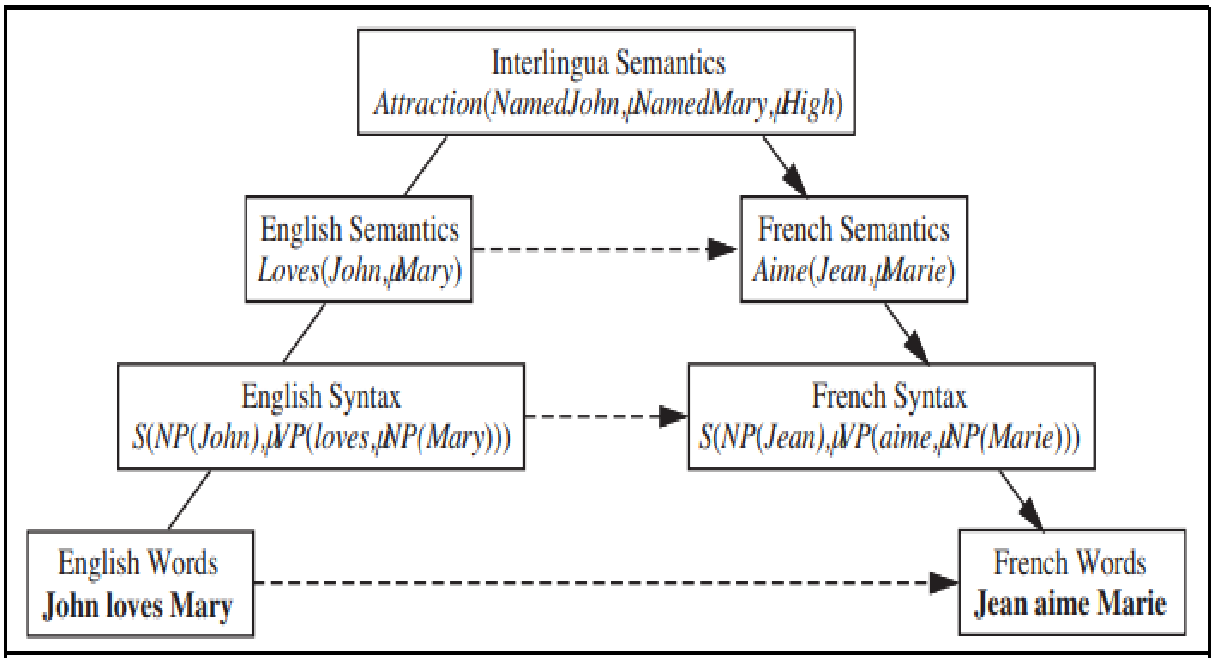
- creating a complete knowledge representation of everything;
- parsing into that representation; and
- generating sentences from that representation.

▪ Other systems are based on a **transfer model**.

▪ They keep a database of translation rules (or examples), and whenever the rule (or example) matches, they translate directly.

▪ Transfer can occur at the lexical, syntactic, or semantic level.

▪ For example, a strictly syntactic rule maps English [*Adjective Noun*] to French [*Noun Adjective*]. A mixed syntactic and lexical rule maps French [S1 "et puis" S2] to English [S1 "and then" S2].



(ii) **Statistical machine translation** :

- Now that we have seen how complex the translation task can be, it should come as no surprise that the most successful machine translation systems are built by training a probabilistic model using statistics gathered from a large corpus of text.
- This approach does not need a complex ontology of interlingua concepts, nor does it need handcrafted grammars of the source and target languages, nor a hand-labeled treebank.
- All it needs is data—sample translations from which a translation model can be learned. To translate a sentence in, say, English (e) into French (f), we find the string of words f ▯ that maximizes

$$f^* = \underset{f}{\operatorname{argmax}}\, P(f \mid e) = \operatorname{argmax} P(e \mid f)\, P(f)\ .$$

- Here the factor P (f) is the target **language model** for French; it says how probable a given sentence is in French. P (e|f) is the **translation mode.**
- All that remains is to learn the phrasal and distortion probabilities. We sketch the procedure;

➔ **Find parallel texts**:

First, gather a parallel bilingual corpus. For example, a **Hansard** is a record of parliamentary debate. Canada, Hong Kong, and other countries produce bilingual Hansards, the European Union publishes its official documents in 11 languages, and the United Nations publishes multilingual documents.

➔ **Segment into sentences**: The unit of translation is a sentence, so we will have to break the corpus into sentences. Periods are strong indicators of the end of a sentence.

➔ **Align sentences**: For each sentence in the English version, determine what sentence(s) it corresponds to in the French version.

Usually, the next sentence of English corresponds to the next sentence of French in a 1:1 match, but sometimes there is variation: one sentence in one language will be split into a 2:1 match, or the order of two sentences will be swapped, resulting in a 2:2 match.

➔ **Align phrases**: Within a sentence, phrases can be aligned by a process that is similar to that used for sentence alignment, but requiring iterative improvement.

➔ **Extract distortions**: Once we have an alignment of phrases we can define distortion probabilities. Simply count how often distortion occurs in the corpus for each distance d = 0, ±1, ±2, . . ., and apply smoothing.

➔ **Improve estimates with EM**: Use expectation–maximization to improve the estimates of P(f | e) and P(d) values.

We compute the best alignments with the current values of these parameters in the E step, then update the estimates in the M step and iterate the process until convergence.

4.1.5. SPEECH RECOGNITION :

- **Speech recognition** is the task of identifying a sequence of words uttered by a speaker, given the acoustic signal.
- It has become one of the mainstream applications of AI—millions of people interact with speech recognition systems every day to navigate voice mail systems, search the Web from mobile phones, and other applications.
- Speech is an attractive option when hands-free operation is necessary, as when operating machinery.

  Speech recognition is difficult because the sounds made by a speaker are ambiguous and, well, noisy.
- Several issues that make speech problematic :
- First, **segmentation**: written words in English have spaces between them, but in fast speech there are no pauses in "wreck a nice" that would distinguish it as a multiword phrase as opposed to the single word "recognize."
- Second, **coarticulation**: when speaking quickly the "s" sound at the end of "nice" merges with the "b" sound at the beginning of "beach," yielding something that is close to a "sp."
- Another problem that does not show up in this example is **homophones**—words like "to," "too," and "two" that sound the same but differ in meaning.
- As usual, the most likely sequence can be computed with the help of Bayes' rule to be:

$$\underset{word_{1:t}}{\operatorname{argmax}} P(word_{1:t} \mid sound_{1:t}) = \underset{word_{1:t}}{\operatorname{argmax}} P(sound_{1:t} \mid word_{1:t}) P(word_{1:t}) \,.$$

- Here P(sound 1:t|word 1:t) is the **acoustic model**. It describes the sounds of words—that "ceiling" begins with a soft "c" and sounds the same as "sealing."

  P(word 1:t) is known as the **language model**. It specifies the prior probability of each utterance—for example, that "ceiling fan" is about 500 times more likely as a word sequence than "sealing fan."
- This approach was named the **noisy channel model** by Claude Shannon (1948).
- He described a situation in which an original message (the *words* in our example) is transmitted over a noisy channel (such as a telephone line) such that a corrupted message (the *sounds* in our example) are received at the other end.
- Once we define the acoustic and language models, we can solve for the most likely sequence of

words using the Viterbi algorithm .

- Most speech recognition systems use a language model that makes the Markov assumption—that the current state Word t depends only on a fixed number n of previous states—and represent Word t as a single random variable taking on a finite set of values, which makes it a Hidden Markov Model (HMM).

  Thus, speech recognition becomes a simple application of the HMM methodology,

(i) **Acoustic model** :

- Sound waves are periodic changes in pressure that propagate through the air. When these waves strike the diaphragm of a microphone, the back-and-forth movement generates an electric current.
- An analog-to-digital converter measures the size of the current—which approximates the amplitude of the sound wave—at discrete intervals called the **sampling rate**.

  Speech sounds, which are mostly in the range of 100 Hz (100 cycles per second) to 1000 Hz, are typically sampled at a rate of 8 kHz. (CDs and mp3 files are sampled at 44.1 kHz.)
- The precision of each measurement is determined by the **quantization factor**; speech recognizers typically keep 8 to 12 bits.
- That means that a low-end system, sampling at 8 kHz with 8-bit quantization, would require nearly half a megabyte per minute of speech.
- Since we only want to know what words were spoken, not exactly what they sounded like, we don't need to keep all that information.
- We only need to distinguish between different speech sounds. Linguists have identified about 100 speech sounds, or **phones**, that can be composed to form all the words in all known human languages.
- Roughly speaking, a phone is the sound that corresponds to a single vowel or consonant, but there are some complications: combinations of letters, such as "th" and "ng" produce single phones, and some letters produce different phones in different contexts all the phones that are used in English.
- A **phoneme** is the smallest unit of sound that has a distinct meaning to speakers of a particular language.
- For example, the "t" in "stick" sounds similar enough to the "t" in "tick" that speakers of English consider them the same phoneme.
- First, we observe that although the sound frequencies in speech may be several kHz, the *changes* in the content of the signal occur much less often, perhaps at no more than 100 Hz.

- Therefore, speech systems summarize the properties of the signal over time slices called **frames**.

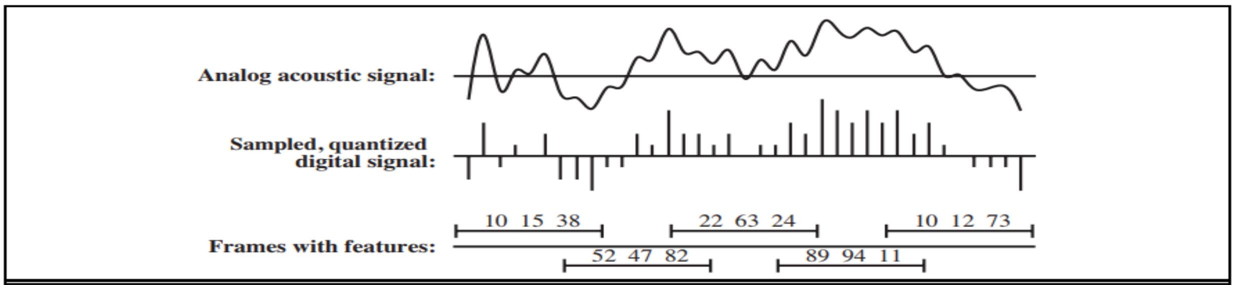


Figure 23.15 Translating the acoustic signal into a sequence of frames. In this diagram each frame is described by the discretized values of three acoustic features; a real system would have dozens of features.

(ii) Language model :

- For general-purpose speech recognition, the language model can be an n-gram model of text learned from a corpus of written sentences.
- However, spoken language has different characteristics than written language, so it is better to get a corpus of transcripts of spoken language.
- For task-specific speech recognition, the corpus should be task-specific: to build your airline reservation system, get transcripts of prior calls.

  It also helps to have task-specific vocabulary, such as a list of all the airports and cities served, and all the flight numbers.

(iii) **Building a speech recognizer** :

- The quality of a speech recognition system depends on the quality of all of its components— the language model, the word-pronunciation models, the phone models, and the signal processing algorithms used to extract spectral features from the acoustic signal.
- The accuracy of a system depends on a number of factors. First, the quality of the signal matters: a high-quality directional microphone aimed at a stationary mouth in a padded room will do much better than a cheap microphone transmitting a signal over phone lines from a car in traffic with the radio playing.

PERCEPTION:

Perception provides agents with information about the world they inhabit by interpreting the response of **sensors**.

A sensor measures some aspect of the environment in a form that can be used as input by an agent program. The sensor could be as simple as a switch, which gives one bit telling whether it is on or off, or as complex as the eye. A variety of sensory modalities are available to artificial agents.

Some robots do **active sensing**, meaning they send out a signal, such as radar or ultrasound, and sense the reflection of this signal off of the environment.

The problem for a vision-capable agent then is: Which aspects of the rich visual stimulus should be considered to help the agent make good action choices, and which aspects should be ignored? Vision—and all perception—serves to further the agent's goals, not as an end to itself.

In the **recognition** approach an agent draws distinctions among the objects it encounters based on visual and other information. Recognition could mean labelling each image with a yes or no as to whether it contains food that we should forage, or contains Grandma's face.

IMAGE FORMATION

Imaging distorts the appearance of objects. For example, a picture taken looking down a long straight set of railway tracks will suggest that the rails converge and meet. As another example, if you hold your hand in front of your eye, you can block out the moon, which is not smaller than your hand. As you move your hand back and forth or tilt it, your hand will seem to shrink and grow in the image, but it is not doing so in reality. Models of these effects are essential for both recognition and reconstruction.

Images without lenses: The pinhole camera

Image sensors gather light scattered from objects in a **scene** and create a two-dimensional **image**. In the eye, the image is formed on the retina, which consists of two types of cells: about 100 million rods, which are sensitive to light at a wide range of wavelengths, and 5 million cones. Cones, which are essential for colour vision, are of three main types, each of which is sensitive to a different set of wavelengths.

In cameras, the image is formed on an image plane, which can be a piece of film coated with silver halides or a rectangular grid of a few million photosensitive **pixels**, each a complementary metal-oxide semiconductor (CMOS) or charge-coupled device (CCD).

Each photon arriving at the sensor produces an effect, whose strength depends on the wavelength of the photon. The output of the sensor is the sum of all effects due to photons observed in some time window, meaning that image sensors report a weighted average of the intensity of light arriving at the sensor.

To see a focused image, we must ensure that all the photons from approximately the same spot in the scene arrive at approximately the same point in the image plane.

The simplest way to form a focused image is to view stationary objects with a **pinhole camera**, which consists of a pinhole opening, O, at the front of a box, and an image plane at the back of the box. Photons from the scene must pass through the pinhole, so if it is small enough then nearby photons in the scene will be nearby in the image plane, and the image will be in focus.

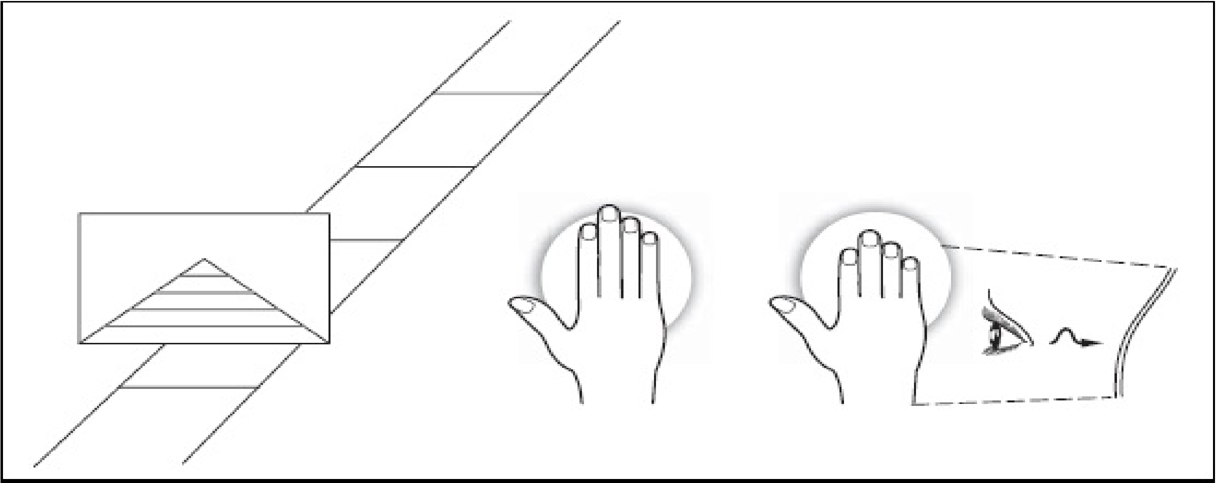
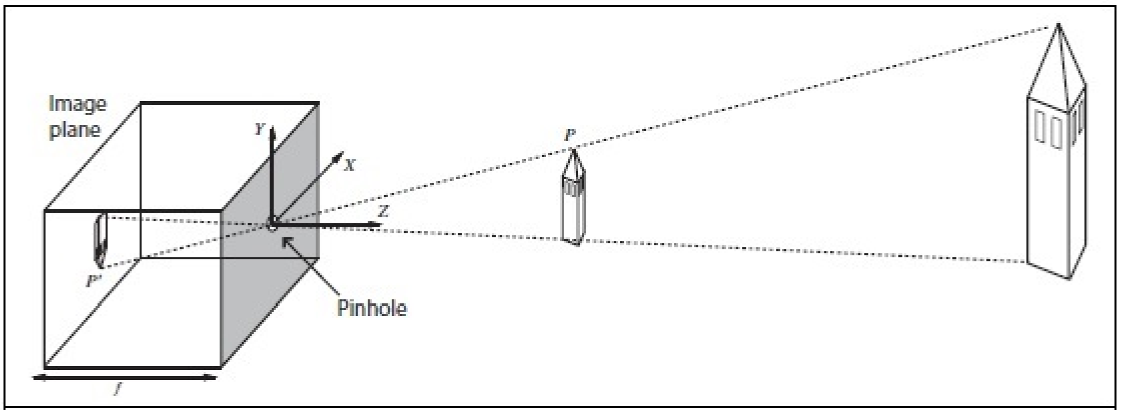


Figure 24.1 Imaging distorts geometry. Parallel lines appear to meet in the distance, as in the image of the railway tracks on the left. In the center, a small hand blocks out most of a large moon. On the right is a foreshortening effect: the hand is tilted away from the eye, making it appear shorter than in the center figure.

The geometry of scene and image is easiest to understand with the pinhole camera. We use a three-dimensional coordinate system with the origin at the pinhole, and consider a point P in the scene, with coordinates (X, Y,Z). P gets projected to the point P'_ in the image plane with coordinates (x, y, z). If f is the distance from the pinhole to the image plane, then by similar triangles, we can derive the following equations.

$$\frac{-x}{f} = \frac{X}{Z}, \; \frac{-y}{f} = \frac{Y}{Z} \quad \Rightarrow \quad x = \frac{-fX}{Z}, \; y = \frac{-fY}{Z} \; .$$

These equations define an image-formation process known as **perspective projection**.



Each light-sensitive element in the image plane at the back of a pinhole camera receives light from a the small range of directions that passes through the pinhole. If the pinhole is small enough, the result is a focused image at the back of the pinhole. The process of projection means that large, distant objects look the same as smaller, nearby objects. Note that the image is projected upside down.

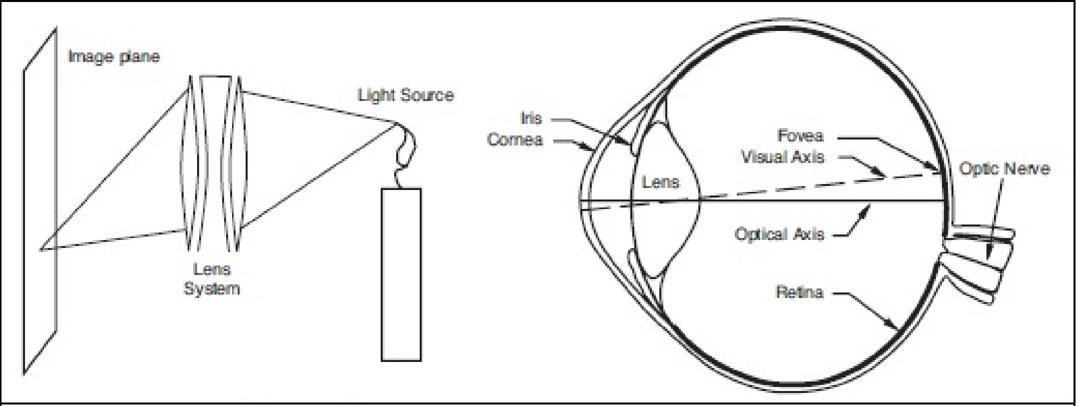
Under perspective projection, distant objects look small. This is what allows you to cover the moon with your hand (Figure 24.1). An important result of this effect is that parallel lines converge to a point on the horizon. (Think of railway tracks, Figure 24.1.) A line in the scene in the direction (U, V,W) and passing through the point (X0, Y0, Z0) can be described as the set of points (X0 + λU, Y0 + λV,Z0 + λW), with λ varying between −∞ and +∞. Different choices of (X0, Y0, Z0) yield different lines parallel to one another. The projection of a point Pλ from this line onto the image plane is given by

$$\left(f\frac{X_0+\lambda U}{Z_0+\lambda W}, f\frac{Y_0+\lambda V}{Z_0+\lambda W}\right)$$

As λ→ ∞or λ → −∞, this becomes p∞ = (fU/W, fV/W) if W _= 0. This means that two parallel lines leaving different points in space will converge in the image—for large λ, the image points are nearly the same, whatever the value of (X0, Y0, Z0) (again, think railway tracks,). We call p∞ the **vanishing point** associated with the family of straight lines with direction (U, V,W). Lines with the same direction share the same vanishing point.

Lens systems

The drawback of the pinhole camera is that we need a small pinhole to keep the image in focus. But the smaller the pinhole, the fewer photons get through, meaning the image will be dark. We can gather more photons by keeping the pinhole open longer, but then we will get **motion blur**—objects in the scene that move will appear blurred because they send photons to multiple locations on the image plane. If we can't keep the pinhole open longer, we can try to make it bigger. More light will enter, but light from a small patch of object in the scene will now be spread over a patch on the image plane, causing a blurred image.



Lenses collect the light leaving a scene point in a range of directions, and steer it all to arrive at a single point on the image plane. Focusing works for points lying close to a focal plane in space; other points will not be focused properly. In cameras, elements of the lens system move to change the focal plane, whereas in the eye, the shape of the lens is changed by specialized muscles.

Vertebrate eyes and modern cameras use a **lens** system to gather sufficient light while keeping the image in focus. A large opening is covered with a lens that focuses light from nearby object locations down to nearby locations in the image plane. However, lens systems have a limited **depth of field**: they can focus light only from points that lie within a range of depths (centered around a **focal plane**). Objects outside this range will be out of focus in the image. To move the focal plane, the lens in the eye can change shape in a camera, the lenses move back and forth.

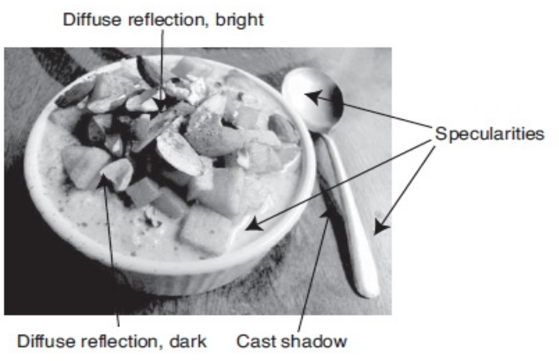
Scaled orthographic projection

Perspective effects aren't always pronounced. For example, spots on a distant leopard may look small because the leopard is far away, but two spots that are next to each other will have about the same size. This is because the difference in distance to the spots is small compared to the distance to them, and so we can simplify the projection model. The appropriate model is **scaled orthographic projection**. The idea is as follows: If the depth Z of points on the object varies within some range Z0 +- ΔZ, with ΔZ * Z0, then the perspective scaling factor f/Z can be approximated by a constant s = f/Z0. The equations for projection from the scene coordinates (X, Y,Z) to the image plane become x = sX and y = sY . Scaled orthographic projection is an approximation that is valid only for those parts of the scene with not much internal depth variation. For example, scaled orthographic projection can be a good model for the features on the front of a distant building.

Light and shading

The brightness of a pixel in the image is a function of the brightness of the surface patch in the scene that projects to the pixel. We will assume a linear model (current cameras have nonlinearities at the extremes of light and dark, but are linear in the middle). Image brightness is a strong, if ambiguous, cue to the shape of an object, and from there to its identity. People are usually able to distinguish the three main causes of varying brightness and reverse-engineer the object's properties.

The first cause is **overall intensity** of the light. Even though a white object in shadow may be less bright than a black object in direct sunlight, the eye can distinguish relative brightness well, and perceive the white object as white. Second, different points in the scene may **reflect** more or less of the light. Third, surface patches facing the light are brighter than surface patches tilted away from the light, an effect known as **shading**. Most surfaces reflect light by a process of **diffuse reflection**. Diffuse reflection scatters light evenly across the directions leaving a surface, so the brightness of a diffuse surface doesn't depend on the viewing direction. The behaviour of a perfect mirror is known as specular **reflection**. Some surfaces—such as brushed metal, plastic, or a wet floor—display small patches where specular reflection has occurred, called **secularities**. These are easy to identify, because they are small and bright. For almost all purposes, it is enough to model all surfaces as being diffuse with secularities.



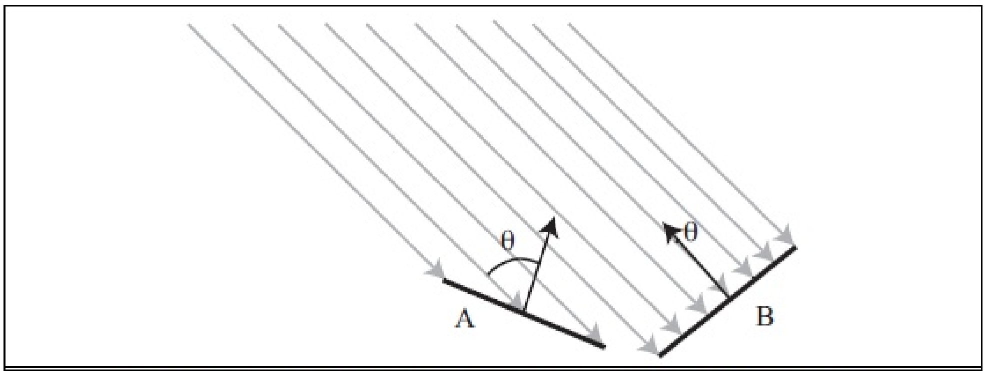
A variety of illumination effects. There are specularities on the metal spoon and on the milk. The bright diffuse surface is bright because it faces the light direction. The dark diffuse surface is dark because it is tangential to the illumination direction. The shadows appear at surface points that cannot see the light source. Photo by Mike Linksvayer (mlinksva on flickr).

The main source of illumination outside is the sun, whose rays all travel parallel to one another. We model this behaviour as a **distant point light source**. This is the most important model of lighting, and is quite effective for indoor scenes as well as outdoor scenes. The amount of light collected by a surface patch in this model depends on the angle θ between the illumination direction and the normal to the surface.

A diffuse surface patch illuminated by a distant point light source will reflect some fraction of the light it collects; this fraction is called the **diffuse albedo**. White paper and snow have a high albedo, about 0.90, whereas flat black velvet and charcoal have a low albedo of about 0.05 (which means that 95% of the incoming light is absorbed within the fibres of the velvet or the pores of the charcoal). **Lambert's cosine law** states that the brightness of a diffuse patch is given by

$$I = \rho I_0 \cos\theta\,.$$

where ρ is the diffuse albedo, I0 is the intensity of the light source and θ is the angle between the light source direction and the surface Lampert's law predicts bright image pixels come from surface patches that face the light directly and dark pixels come from patches that see the light only tangentially, so that the shading on a surface provides some shape information. If the surface is not reached by the light source, then it is in **shadow**. Shadows are very seldom a uniform black, because the shadowed surface receives some light from other sources. Outdoors, the most important such source is the sky, which is quite bright. Indoors, light reflected from other S surfaces illuminates shadowed patches. These **interreflections** can have a significant effect on the brightness of other surfaces, too. These effects are sometimes modelled by adding a constant **ambient illumination** term to the predicted intensity.



Two surface patches are illuminated by a distant point source, whose rays are shown as gray arrowheads. Patch A is tilted away from the source (θ is close to 90^0) and collects less energy, because it cuts fewer light rays per unit surface area. Patch B, facing the source (θ is close to 0^0), collects more energy.

Colour

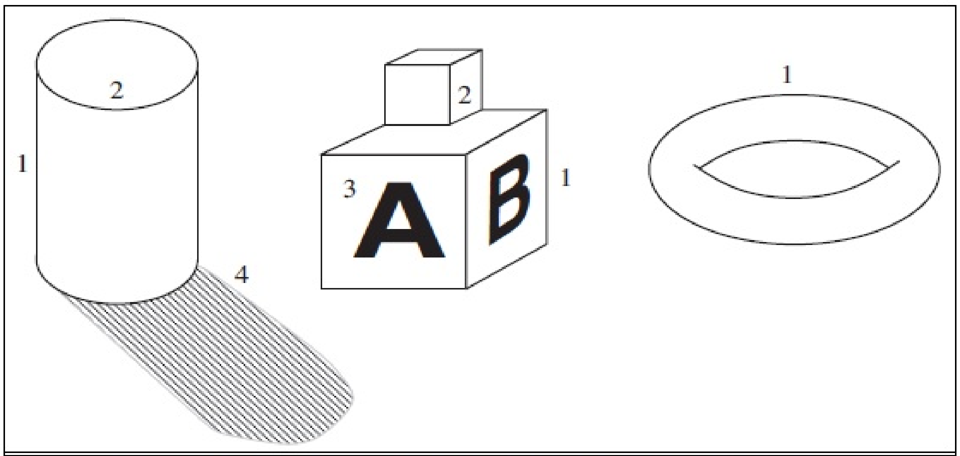
Fruit is a bribe that a tree offers to animals to carry its seeds around. Trees have evolved to have fruit that turns red or yellow when ripe, and animals have evolved to detect these colour changes. Light arriving at the eye has different amounts of energy at different wavelengths; this can be represented by a spectral energy density function. Human eyes respond to light in the 380–750nm wavelength region, with three different types of colour receptor cells, which have peak receptiveness at 420mm (blue), 540nm (green), and 570nm (red). The human eye can capture only a small fraction of the full spectral energy density function—but it is enough to tell when the fruit is ripe.

The **principle of trichromacy** states that for any spectral energy density, no matter how complicated, it is possible to construct another spectral energy density consisting of a mixture of just three colours—usually red, green, and blue—such that a human can't tell the difference between the two. That means that our TVs and computer displays can get by with just the three red/green/blue (or R/G/B) colour elements. It makes our computer vision algorithms easier, too. Each surface can be modelled with three different albedos for R/G/B. Similarly, each light source can be modelled with three R/G/B intensities. We then apply Lambert’s cosine law to each to get three R/G/B pixel values. This model predicts, correctly, that the same surface will produce different coloured image patches under different-coloured lights. In fact, human observers are quite good at ignoring the effects of different coloured lights and are able to estimate the colour of the surface under white light, an effect known as **colour constancy**. Quite accurate colour constancy algorithms are now available; simple versions show up in the “auto white balance” function of your camera. Note that if we wanted to build a camera for mantis shrimp, we would need 12 different pixel colours, corresponding to the 12 types of colour receptors of the crustacean.

EARLY IMAGE-PROCESSING OPERATIONS

We have seen how light reflects off objects in the scene to form an image consisting of, say, five million 3-byte pixels. With all sensors there will be noise in the image, and in any case there is a lot of data to deal with. So how do we get started on analysing this data?

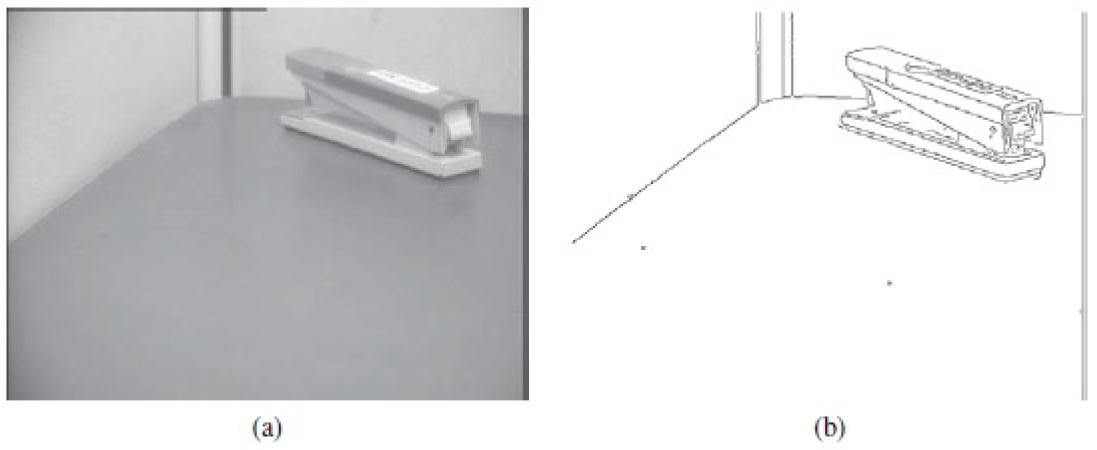
we will study three useful image-processing operations: edge detection, texture analysis, and computation of optical flow. These are called “early” or “low-level” operations because they are the first in a pipeline of operations. Early vision operations are characterized by their local nature (they can be carried out in one part of the image without regard for anything more than a few pixels away) and by their lack of knowledge: we can perform these operations without consideration of the objects that might be present in the scene. This makes the low-level operations good candidates for implementation in parallel hardware—either in a graphics processor unit (GPU) or an eye. We will then look at one mid-level operation: segmenting the image into regions.



Different kinds of edges: (1) depth discontinuities; (2) surface orientation discontinuities; (3) reflectance discontinuities; (4) illumination discontinuities (shadows).

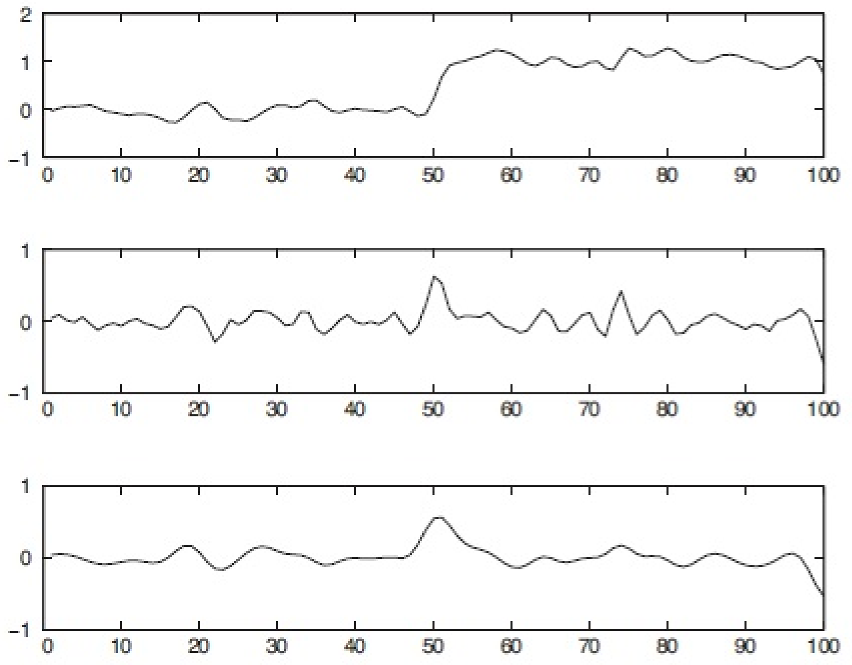
Edge detection

Edges are straight lines or curves EDGE in the image plane across which there is a “significant” change in image brightness. The goal of edge detection is to abstract away from the messy, multimega byte image and toward a more compact, abstract representation, the motivation is that edge contours in the image correspond to important scene contours. In the figure we have three examples of depth discontinuity, labelled 1; two surface-normal discontinuities, labelled 2; a reflectance discontinuity, labelled 3; and an illumination discontinuity (shadow), labelled 4. Edge detection is concerned only with the image, and thus does not distinguish between these different types of scene discontinuities; later processing will.



(a) Photograph of a stapler. (b) Edges computed from (a).

(a) shows an image of a scene containing a stapler resting on a desk, and
(b) shows the output of an edge-detection algorithm on this image. As you can see, there is a difference between the output and an ideal line drawing. There are gaps where no edge appears, and there are “noise” edges that do not correspond to anything of significance in the scene. Later stages of processing will have to correct for these errors.



Top: Intensity profile $I(x)$ along a one-dimensional section across an edge at $x = 50$. Middle: The derivative of intensity, $I'(x)$. Large values of this function correspond to edges, but the function is noisy. Bottom: The derivative of a smoothed version of the intensity, $(I * G_\sigma)'$, which can be computed in one step as the convolution $I * G'_\sigma$. The noisy candidate edge at $x = 75$ has disappeared.

How do we detect edges in an image? Consider the profile of image brightness along a one-dimensional cross-section perpendicular to an edge—for example, the one between the left edge of the desk and the wall. Edges correspond to locations in images where the brightness undergoes a sharp change, so a naive idea would be to differentiate the image and look for places where the magnitude of the derivative I'(x) is large.

One good answer is a weighted average that weights the nearest pixels the most, then gradually decreases the weight for more distant pixels. The **Gaussian filter** does just that. (Users of Photoshop recognize this as the *Gaussian blur* operation.) Recall that the Gaussian function with standard deviation σ and mean 0 is

$$N_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2\sigma^2} \qquad \text{in one dimension, or}$$
$$N_\sigma(x, y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \qquad \text{in two dimensions.}$$

The application of the Gaussian filter replaces the intensity I(x0, y0) with the sum, overall (x, y) pixels, of I(x, y)Nσ(d), where d is the distance from (x0, y0) to (x, y). This kind of weighted sum is so common that there is a special name and notation for it. We say that the function h is the **convolution** of two functions f and g (denoted f ⍰ g) if we have

$$h(x) = (f * g)(x) = \sum_{u=-\infty}^{+\infty} f(u)\, g(x-u) \qquad \text{in one dimension, or}$$

$$h(x,y) = (f * g)(x,y) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} f(u,v)\, g(x-u, y-v) \qquad \text{in two.}$$

There is a natural generalization of this algorithm from one-dimensional cross sections to general two-dimensional images. In two dimensions edges may be at any angle θ. Considering the image brightness as a scalar function of the variables x, y, its gradient is a vector

$$\nabla I = \begin{pmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{pmatrix} = \begin{pmatrix} I_x \\ I_y \end{pmatrix}$$

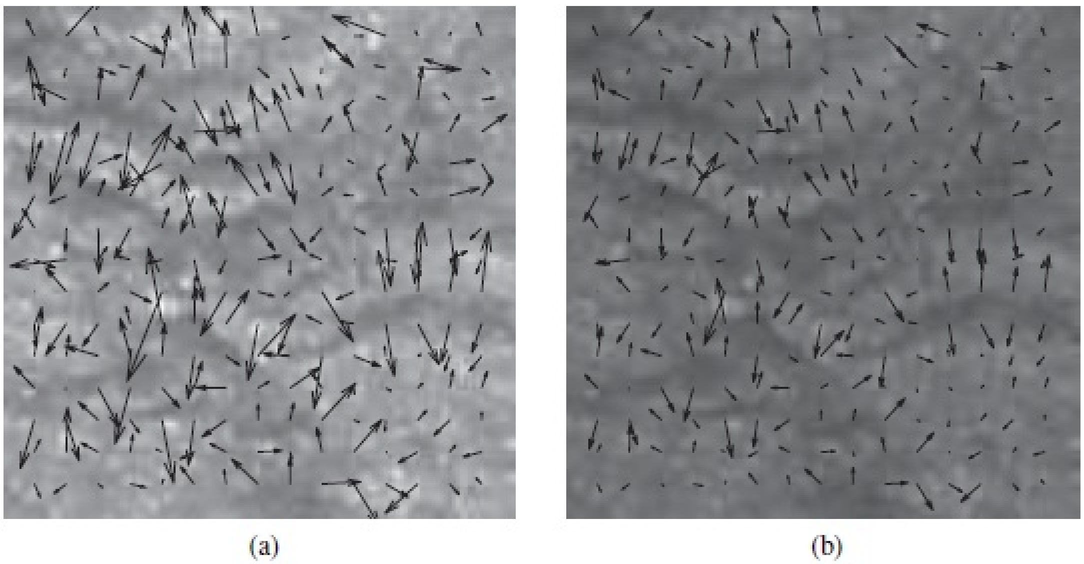
This gives us a θ = θ(x, y) at every pixel, which defines the edge **orientation** at that pixel.

$$\frac{\nabla I}{\|\nabla I\|} = \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}$$

Once we have marked edge pixels by this algorithm, the next stage is to link those pixels that belong to the same edge curves. This can be done by assuming that any two neighbouring edge pixels with consistent orientations must belong to the same edge curve.

Texture

In everyday language, **texture** is the visual TEXTURE feel of a surface—what you see evokes what the surface might feel like if you touched it (“texture” has the same root as “textile”). In computational vision, texture refers to a spatially repeating pattern on a surface that can be sensed visually. Examples include the pattern of windows on a building, stitches on a sweater, spots on a leopard, blades of grass on a lawn, pebbles on a beach, and people in a stadium. Sometimes the arrangement is quite periodic, as in the stitches on a sweater; in other cases, such as pebbles on a beach, the regularity is only statistical.



Two images of the same texture of crumpled rice paper, with different illumination levels. The gradient vector field (at every eighth pixel) is plotted on top of each one. Notice that, as the light gets darker, all the gradient vectors get shorter. The vectors do not rotate, so the gradient orientations do not change.

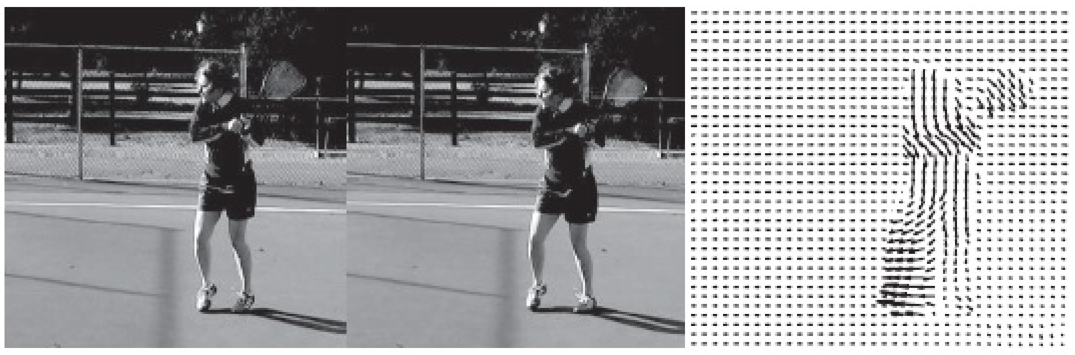
In images of textured objects, edge detection does not work as well as it does for smooth objects. This is because the most important edges can be lost among the texture elements. Quite literally, we may miss the tiger for the stripes. The solution is to look for differences in texture properties, just the way we look for differences in brightness. A patch on a tiger and a patch on the grassy background will have very different orientation histograms, allowing us to find the boundary curve between them.

Optical flow

Next, let us consider what happens when we have a video sequence, instead of just a single static image. When an object in the video is moving, or when the camera is moving relative to an object, the resulting apparent motion in the image is called **optical flow**. Optical flow describes the direction and speed of motion of features in the image—the optical flow of a video of a race car would be measured in pixels per second, not miles per hour. The optical flow encodes useful information about scene structure. For example, in a video of scenery taken from a moving train, distant objects have slower apparent motion than close objects; thus, the rate of apparent motion can tell us something about distance. Optical flow also enables us to recognize actions.

This block of pixels is to be compared with pixel blocks cantered at various candidate pixels at (x_0 + D_x, y_0 + D_y) at time t_0 + D_t. One possible measure of similarity is the **sum of squared differences** (SSD):

$$\mathrm{SSD}(D_x, D_y) = \sum_{(x,y)} (I(x,y,t) - I(x+D_x, y+D_y, t+D_t))^2 .$$

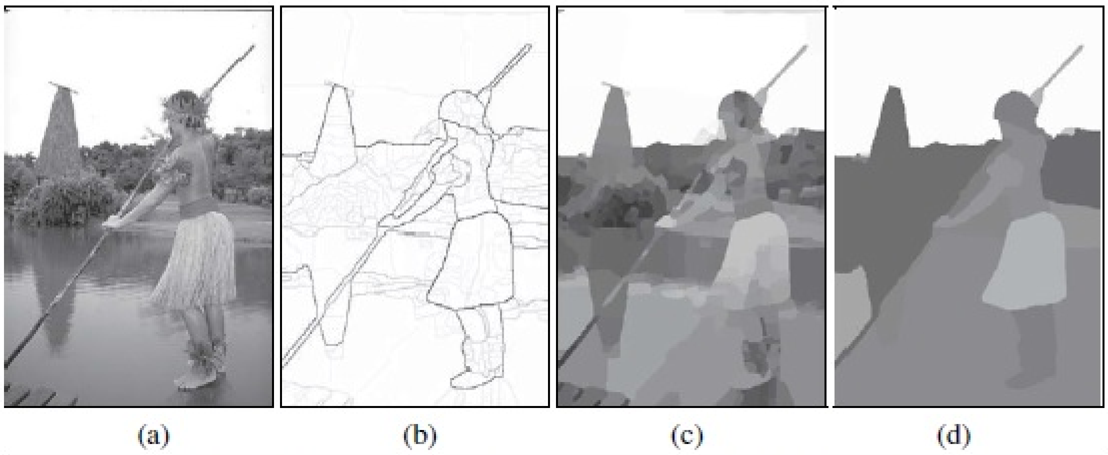


Two frames of a video sequence. On the right is the optical flow field corresponding to the displacement from one frame to the other. Note how the movement of the tennis racket and the front leg is captured by the directions of the arrows. (Courtesy of Thomas Brox.)

Segmentation of images

Segmentation is the process of breaking an image into **regions** of similar pixels. Each image pixel can be associated with certain visual properties, such as brightness, colour, and texture. Within an object, or a single part of an object, these attributes vary relatively little, whereas across an inter-object boundary there is typically a large change in one or more of these attributes.

Boundaries detected by this technique turn out to be significantly better than those found using the simple edge-detection technique described previously. But still there are two limitations. (1) The boundary pixels formed by thresholding Pb(x, y, θ) are not guaranteed to form closed curves, so this approach doesn't deliver regions, and (2) the decision making exploits only local context and does not use global consistency constraints.



(a) Original image. (b) Boundary contours, where the higher the P_b value, the darker the contour. (c) Segmentation into regions, corresponding to a fine partition of the image. Regions are rendered in their mean colors. (d) Segmentation into regions, corresponding to a coarser partition of the image, resulting in fewer regions. (Courtesy of Pablo Arbelaez, Michael Maire, Charles Fowlkes, and Jitendra Malik)

Segmentation based purely on low-level, local attributes such as brightness and colour cannot be expected to deliver the final correct boundaries of all the objects in the scene. To reliably find object boundaries we need high-level knowledge of the likely kinds of objects in the scene. Representing this knowledge is a topic of active research. A popular strategy is to produce an over-segmentation of an image, containing hundreds of homogeneous regions known as **super pixels**. From there, knowledge-based algorithms can take over; they will find it easier to deal with hundreds of super pixels rather than millions of raw pixels. How to exploit high-level knowledge of objects is the subject of the next section.

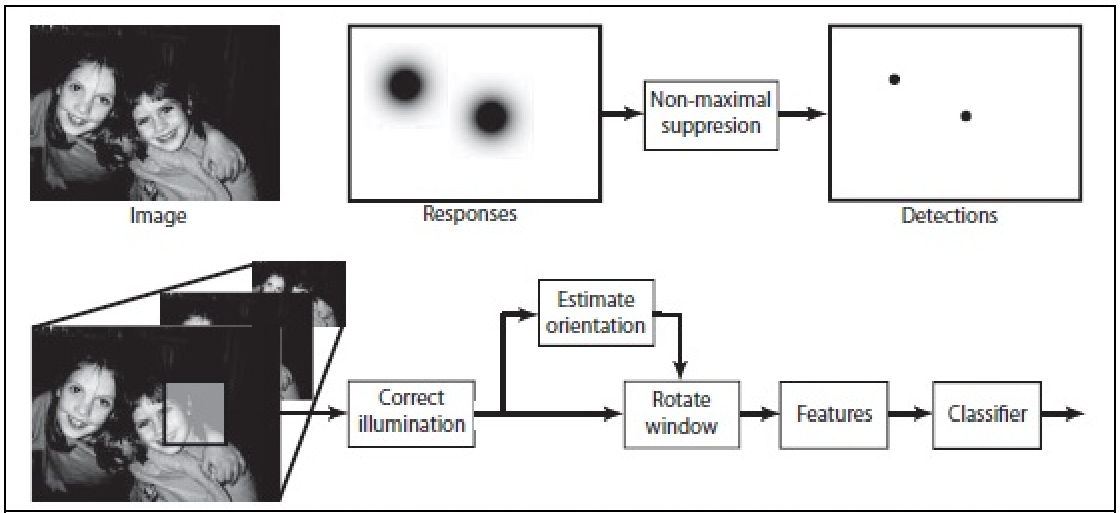
OBJECT RECOGNITION BY APPEARANCE

Appearance is shorthand for what an object tends to look like. Some object categories—for example, baseballs—vary rather little in appearance; all of the objects in the category look about the same under most circumstances. In this case, we can compute a set of features describing each class of images likely to contain the object, then test it with a classifier.

Testing each class of images with a learned classifier is an important general recipe. It works extremely well for faces looking directly at the camera, because at low resolution and under reasonable lighting, all such faces look quite similar. The face is round, and quite bright compared to the eye sockets; these are dark, because they are sunken, and the mouth is a dark slash, as are the eyebrows. Major changes of illumination can cause some variations in this pattern, but the range of variation is quite manageable. That makes it possible to detect face positions in an image that contains faces. Once a computational challenge, this feature is now commonplace in even inexpensive digital cameras.

For the moment, we will consider only faces where the nose is oriented vertically; we will deal with rotated faces below. We sweep a round window of fixed size over the image, compute features for it, and present the features to a classifier. This strategy is sometimes called the **sliding window**. Features need to be robust to shadows and to changes in brightness caused by illumination changes. One strategy is to build features out of gradient orientations. Another is to estimate and correct the illumination in each image window. To find faces of different sizes, repeat the sweep over larger or smaller versions of the image. Finally, we postprocess the responses across scales and locations to produce the final set of detections.

Training data is quite easily obtained. There are several data sets of marked-up face images, and rotated face windows are easy to build (just rotate a window from a training data set). One trick that is widely used is to take each example window, then produce new examples by changing the orientation of the window, the center of the window, or the scale very slightly. This is an easy way of getting a bigger data set that reflects real images fairly well; the trick usually improves performance significantly.

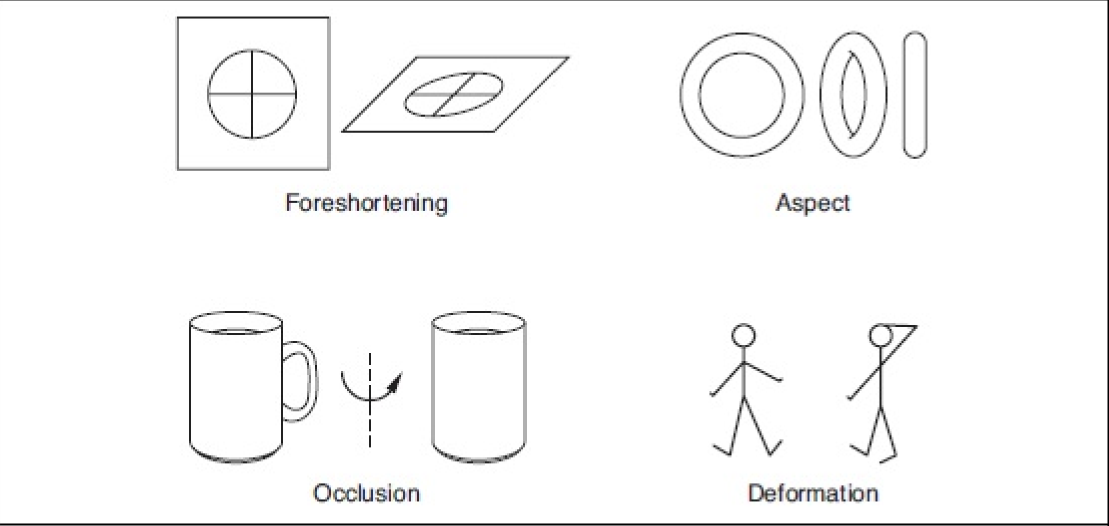


Face finding systems vary, but most follow the architecture illustrated in two parts here. On the top, we go from images to responses, then apply non-maximum suppression to find the strongest local response. The responses are obtained by the process illustrated on the bottom. We sweep a window of fixed size over larger and smaller versions of the image, so as to find smaller or larger faces, respectively. The illumination in the window is corrected, and then a regression engine (quite often, a neural net) predicts the orientation of the face. The window is corrected to this orientation and then presented to a classifier. Classifier outputs are then postprocessed to ensure that only one face is placed at each location in the image.

Complex appearance and pattern elements

Many objects produce much more complex patterns than faces do. This is because several effects can move features around in an image of the object.

- **Foreshortening**, which causes a pattern viewed at a slant to be significantly distorted.
- **Aspect**, which causes objects to look different when seen from different directions. Even as simple an object as a doughnut has several aspects; seen from the side, it looks like a flattened oval, but from above it is an annulus.
- **Occlusion**, where some parts are hidden from some viewing directions. Objects can occlude one another, or parts of an object can occlude other parts, an effect known as self-occlusion.
- **Deformation**, where internal degrees of freedom of the object change its appearance. For example, people can move their arms and legs around, generating a very wide range of different body configurations.

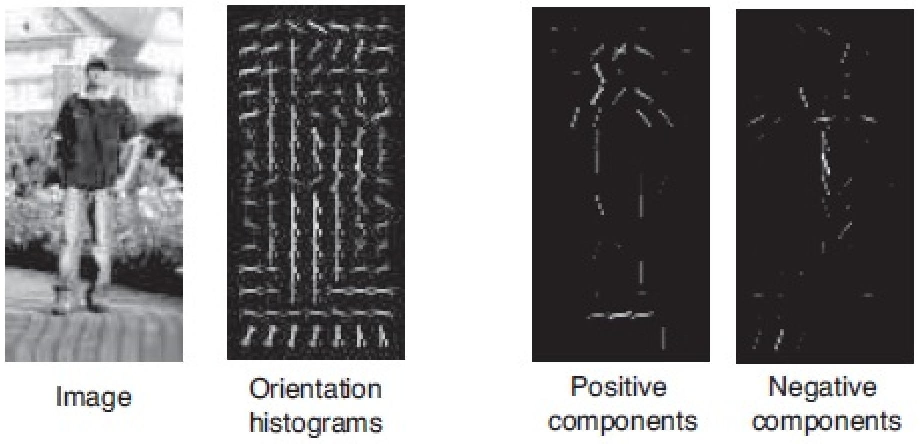


Sources of appearance variation. First, elements can foreshorten, like the circular patch on the top left. This patch is viewed at a slant, and so is elliptical in the image. Second, objects viewed from different directions can change shape quite dramatically, a phenomenon known as aspect. On the top right are three different aspects of a doughnut. Occlusion causes the handle of the mug on the bottom left to disappear when the mug is rotated. In this case, because the body and handle belong to the same mug, we have self-occlusion. Finally, on the bottom right, some objects can deform dramatically.

The most obvious approach is to represent the image window with a histogram of the pattern elements that appear there. This approach does not work particularly well, because too many patterns get confused with one another. For example, if the pattern elements are color pixels, the French, UK, and Netherlands flags will get confused because they have approximately the same color histograms, though the colors are arranged in very different ways. Quite simple modifications of histograms yield very useful features. The trick is to preserve some spatial detail in the representation; for example, headlights tend to be at the front of a car and wheels tend to be at the bottom. Histogram-based features have been successful in a wide variety of recognition applications; we will survey pedestrian detection.

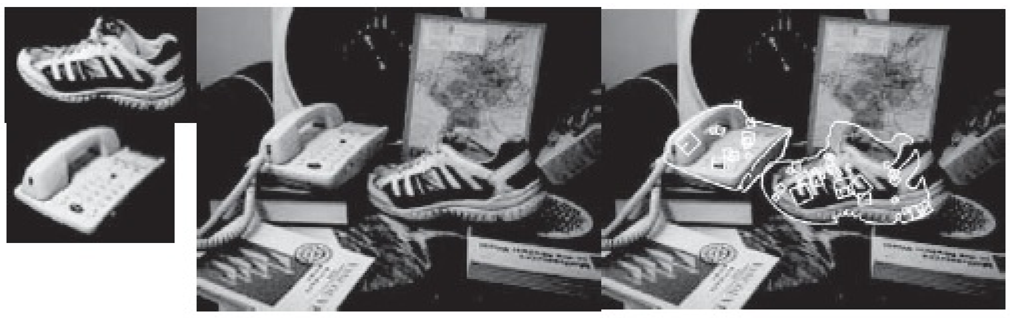
Pedestrian detection with HOG features

The World Bank estimates that each year car accidents kill about 1.2 million people, of whom about two thirds are pedestrians. This means that detecting pedestrians is an important application problem, because cars that can automatically detect and avoid pedestrians might save many lives. Pedestrians wear many different kinds of clothing and appear in many different configurations, but, at relatively low resolution, pedestrians can have a fairly characteristic appearance. The most usual cases are lateral or frontal views of a walk. In these cases, we see either a “lollipop” shape — the torso is wider than the legs, which are together in the stance phase of the walk — or a “scissor” shape — where the legs are swinging in the walk. We expect to see some evidence of arms and legs, and the curve around the shoulders and head also tends to visible and quite distinctive. This means that, with a careful feature construction, we can build a useful moving-window pedestrian detector.



Local orientation histograms are a powerful feature for recognizing even quite complex objects. On the left, an image of a pedestrian. On the center left, local orientation histograms for patches. We then apply a classifier such as a support vector machine to find the weights for each histogram that best separate the positive examples of pedestrians from non-pedestrians. We see that the positively weighted components look like the outline of a person. The negative components are less clear; they represent all the patterns that are not pedestrians. Figure from Dalal and Triggs (2005) © IEEE.

One further trick is required to make a good feature. Because orientation features are not affected by illumination brightness, we cannot treat high-contrast edges specially. This means that the distinctive curves on the boundary of a pedestrian are treated in the same way as fine texture detail in clothing or in the background, and so the signal may be submerged in noise. We can recover contrast information by counting gradient orientations with weights that reflect how significant a gradient is compared to other gradients in the same cell.



Another example of object recognition, this one using the SIFT feature (Scale Invariant Feature Transform), an earlier version of the HOG feature. On the **left**, images of a shoe and a telephone that serve as object models. In the **center**, a test image. On the **right**, the shoe and the telephone have been detected by: finding points in the image whose SIFT feature descriptions match a model; computing an estimate of pose of the model; and verifying that estimate. A strong match is usually verified with rare false positives. Images from Lowe (1999) © IEEE.

orientation at **x** for this cell. A natural choice of weight is

$$w_{\mathbf{x},C} = \frac{\|\nabla I_{\mathbf{x}}\|}{\sum_{\mathbf{u}\in C}\|\nabla I_{\mathbf{u}}\|} .$$

This compares the gradient magnitude to others in the cell, so gradients that are large compared to their neighbours get a large weight. The resulting feature is usually called a **HOG feature**

This feature construction is the main way in which pedestrian detection differs from face detection. Otherwise, building a pedestrian detector is very like building a face detector. The detector sweeps a window across the image, computes features for that window, then presents it to a classifier. Non-maximum suppression needs to be applied to the output. In most applications, the scale and orientation of typical pedestrians is known. For example, in driving applications in which a camera is fixed to the car, we expect to view mainly vertical pedestrians, and we are interested only in nearby pedestrians. Several pedestrian data sets have been published, and these can be used for training the classifier.

RECONSTRUCTING THE 3D WORLD

In this section we show how to go from the two-dimensional image to a three-dimensional representation of the scene. The fundamental question is this: Given that all points in the scene that fall along a ray to the pinhole are projected to the same point in the image, how do we recover three-dimensional information? Two ideas come to our rescue

- If we have two (or more) images from different camera positions, then we can triangulate to find the position of a point in the scene.
- We can exploit background knowledge about the physical scene that gave rise to the image. Given an object model **P**(Scene) and a rendering model **P**(Image | Scene), we can compute a posterior distribution **P**(Scene | Image).

There is as yet no single unified theory for scene reconstruction. We survey eight commonly used visual cues: **motion**, **binocular stereopsis**, **multiple views**, **texture**, **shading**, **contour**, and **familiar objects**.

Motion parallax

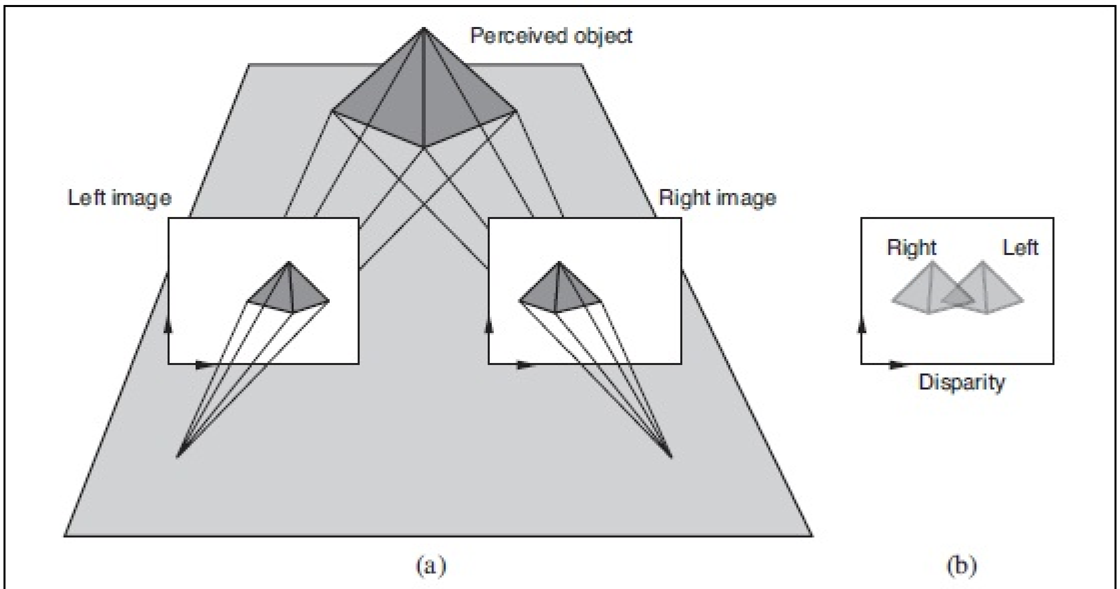
If the camera moves relative to the three-dimensional scene, the resulting apparent motion in the image, optical flow, can be a source of information for both the movement of the camera and depth in the scene. To understand this, we state (without proof) an equation that relates the optical flow to the viewer’s translational velocity **T** and the depth in the scene. The components of the optical flow field are

$$v_x(x,y) = \frac{-T_x + xT_z}{Z(x,y)}, \qquad v_y(x,y) = \frac{-T_y + yT_z}{Z(x,y)} ,$$

where Z(x, y) is the z-coordinate of the point in the scene corresponding to the point in the image at (x, y).

Note that both components of the optical flow, $v_x(x, y)$ and $v_y(x, y)$, are zero at the point $x = T_x/T_z, y = T_y/T_z$. This point is called the **focus of expansion** of the flow field. Suppose we change the origin in the x–y plane to lie at the focus of expansion; then the expressions for optical flow take on a particularly simple form. Let (x', y') be the new coordinates defined by $x' = x - T_x/T_z$, $y' = y - T_y/T_z$. Then

$$v_x(x', y') = \frac{x'T_z}{Z(x', y')}, \qquad v_y(x', y') = \frac{y'T_z}{Z(x', y')} .$$



Translating a camera parallel to the image plane causes image features to move in the camera plane. The disparity in positions that results is a cue to depth. If we superimpose left and right image, as in (b), we see the disparity.

OBJECT RECOGNITION FROM STRUCTURAL INFORMATION

Putting a box around pedestrians in an image may well be enough to avoid driving into them. We have seen that we can find a box by pooling the evidence provided by orientations, using histogram methods to suppress potentially confusing spatial detail. If we want to know more about what someone is doing, we will need to know where their arms, legs, body, and head lie in the picture. Individual body parts are quite difficult to detect on their own using a moving window method, because their color and texture can vary widely and because they are usually small in images. Often, forearms and shins are as small as two to three pixels wide. Body parts do not usually appear on their own, and representing what is connected to what could be quite powerful, because parts that are easy to find might tell us where to look for parts that are small and hard to detect.

Inferring the layout of human bodies in pictures is an important task in vision, because the layout of the body often reveals what people are doing. A model called a **deformable template** can tell us which configurations are acceptable: the elbow can bend but the head is never joined to the foot. The simplest deformable template model of a person connects lower arms to upper arms, upper arms to the torso, and so on.

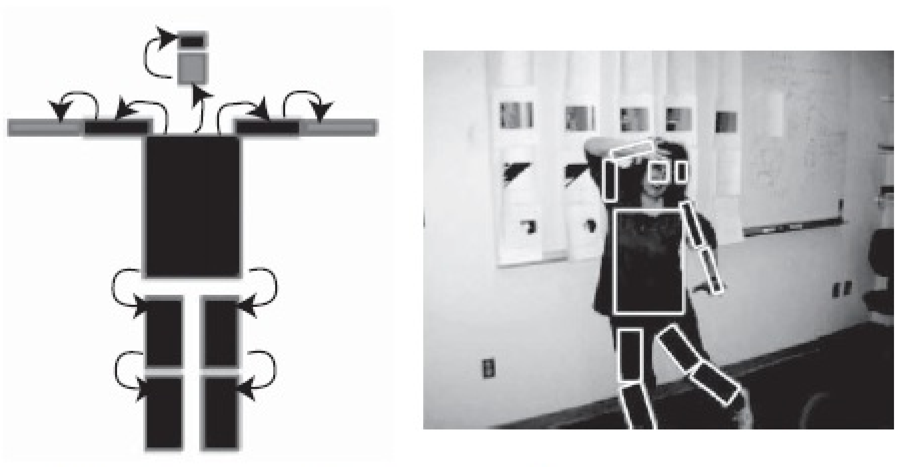
we could represent the fact that left and right upper arms tend to have the same color and texture, as do left and right legs.

The geometry of bodies: Finding arms and legs

For the moment, we assume that we know what the person's body parts look like (e.g., we know the color and texture of the person's clothing). We can model the geometry of the body as a tree of eleven segments (upper and lower left and right arms and legs respectively, a torso, a face, and hair on top of the face) each of which is rectangular. We assume that the position and orientation (**pose**) of POSE the left lower arm is independent of all other segments given the pose of the left upper arm; that the pose of the left upper arm is independent of all segments given the pose of the torso; and extend these assumptions in the obvious way to include the right arm and the legs, the face, and the hair. Such models are often called "cardboard people" models.

There are two criteria for evaluating a configuration. First, an image rectangle should look like its segment. For the moment, we will remain vague about precisely what that means, but we assume we have a function φi that scores how well an image rectangle matches a body segment. For each pair of related segments, we have another function ψ that scores how well relations between a pair of image rectangles match those to be expected from the body segments. The dependencies between segments form a tree, so each segment has only one parent, and we could write ψi,pa(i).

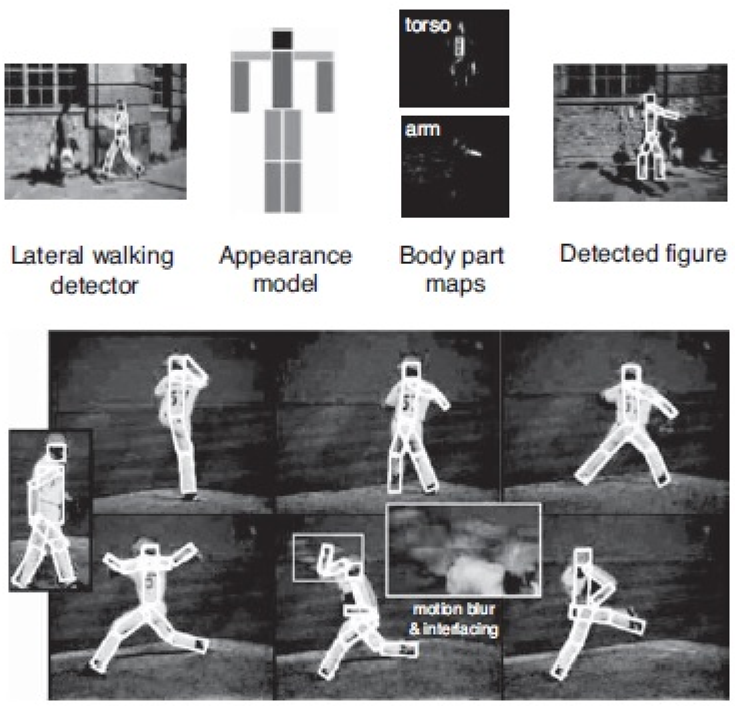
$$\sum_{i \in \mathbf{segments}} \phi_i(m_i) + \sum_{i \in \mathbf{segments}} \psi_{i,\mathbf{pa}(i)}(m_i, m_{\mathbf{pa}(i)}) \ .$$



A pictorial structure model evaluates a match between a set of image rectangles and a cardboard person (shown on the left) by scoring the similarity in appearance between body segments and image segments and the spatial relations between the image segments. Generally, a match is better if the image segments have about the right appearance and are in about the right place with respect to one another. The appearance model uses average colors for hair, head, torso, and upper and lower arms and legs. The relevant relations are shown as arrows. On the right, the best match for a particular image, obtained using dynamic programming. The match is a fair estimate of the configuration of the body. Figure from Felzenszwalb and Huttenlocher (2000) © IEEE.

Coherent appearance: Tracking people in video

Tracking people in video is an important practical problem. If we could reliably report the location of arms, legs, torso, and head in video sequences, we could build much improved game interfaces and surveillance systems. Filtering methods have not had much success with this problem, because people can produce large accelerations and move quite fast. This means that for 30 Hz video, the configuration of the body in frame i doesn't constrain the configuration of the body in frame i+1 all that strongly. Currently, the most effective methods exploit the fact that appearance changes very slowly from frame to frame. If we can infer an appearance model of an individual from the video, then we can use this information in a pictorial structure model to detect that person in each frame of the video. We can then link these locations across time to make a track.



We can track moving people with a pictorial structure model by first obtaining an appearance model, then applying it. To obtain the appearance model, we scan the image to find a lateral walking pose. The detector does not need to be very accurate, but should produce few false positives. From the detector response, we can read off pixels that lie on each body segment, and others that do not lie on that segment. This makes it possible to build a discriminative model of the appearance of each body part, and these are tied together into a pictorial structure model of the person being tracked. Finally, we can reliably track by detecting this model in each frame. As the frames in the lower part of the image suggest, this procedure can track complicated, fast-changing body configurations, despite degradation of the video signal due to motion blur. Figure from Ramanan *et al.* (2007) © IEEE.

USING VISION

Some problems are well understood. If people are relatively small in the video frame, and the background is stable, it is easy to detect the people by subtracting a background image from the current frame. If the absolute value of the difference is large, this **background subtraction** declares the pixel to be a foreground pixel; by linking foreground blobs over time, we obtain a track.

Words and pictures

Many Web sites offer collections of images for viewing. How can we find the images we want? Let's suppose the user enters a text query, such as "bicycle race." Some of the images will have keywords or captions attached, or will come from Web pages that contain text near the image.

In the most straightforward version of this task, we have a set of correctly tagged example images, and we wish to tag some test images. This problem is sometimes known as auto-annotation. The most accurate solutions are obtained using nearest-neighbours' methods. One finds the training images that are closest to the test image in a feature space metric that is trained using examples, then reports their tags.