Automatic programming of machine vision systems

L. J. de Haas

Philips Research Laboratories, PO BOX 80000 Eindhoven, The Netherlands

Abstract.

A system is described which can generate programs for machine vision systems whicji have to measure a number of parameters of an industrial object in a camera image. Programs are generated starting from descriptive object models. The object models used are context-free attribute grammars, hence the generated programs are parsers. Errors in generated programs, caused by using inaccurate models, are screened by comparing the measurements produced by generated programs with values of the desired parameters provided for example images.

Introduction

This paper describes our research on a method to generate programs for machine vision systems. The program generation process is initiated by a query from the user. This query typically requests the system to provide a program that can decide about the presence of a specific object in an image and provide measurements of a number of its parameters (e.g. its position and orientation, hinge openings etc.). In the program generation process we use image generation models, i.e. image grammars. We try to keep these models simple and valid for different classes of scenes by including program testing in example images in the program generation process.

In general the design of an industrial machine vision program depends on a number of factors:

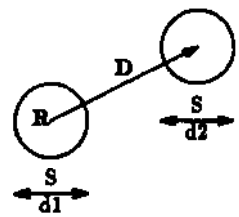
1. The kind of object that must be recognized.
2. The required parameter measurements and a priori available parameters.
3. The available hardware and software to perform primitive measurements on images.
4. The kind of scene the object is in.
5. Overall program specifications, such as speed and reliability

In most current machine vision applications a human programmer accounts for these factors when he writes the program. In this paper we will discuss our research work on the automation of this programming process.

One of the fundamental problems in this automation effort is the fourth factor listed above. This factor subsumes such things as overall image quality, illumination, the presence or absence of confusing objects etc. These factors differ from application to application and their consequences are hard to model in advance. They force the human programmer to write his programs in terms of fairly low-level primitives and to design his programs by trial and error testing in images. We believe that an effective way to avoid having to make detailed application-specific models in advance is to incorporate this trial and error testing of programs in our system.

The design of our systetm is centered around a planner, which, given a model and a goal measurement, generates a sequence of program steps. These steps perform primitive measurements in the image and combine the results in arithmetic steps to produce the goal measurement. The steps of such a program can be divided into two stages, hypothesis generation and verification. In hypothesis generation the program produces the desired measurements and performs only those image observations strictly needed for these measurements. In the verification stage the program uses measured parameters to make predictions that are tested by means of observations in the image. In this paper we will concentrate on the planning of hypothesis generation programs.

The mechanism of our system will be illustrated throughout this paper by means of an example object consisting of two disks of identical size



This "twodisks" object has three parameters: R, the position of one disk, D, the connecting vector to the other disk, and S, the site of the two disks. We will suppose that D has to be measured. We will assume that there is a primitive program "diskprograml". When called by dl—diskprograml () this program produces measurements for the position R and scale S of single disks in the image. These are then available as dI.R and dI.S respectively. (Note that "diskprograml ()" does not produce a unique result if there is more than one disk in the image. Hence we presuppose some mechanism to go through alternative results, one pair R,S for every disk in the image, e.g. by means of backtracking). A typical program to produce D in a hypothetical twodisk object in an image is:

1. **d1 ← diskprogram1()**
2. **d2 ← diskprogram1()**
3. **D ← d2.R - d1.R**

This hypothesis can be verified further by the following verification program:

D ≠ 0
d1.S ← d2.S

Our planner accounts for the first three factors influencing programs by means of a number of input sources:

1. object models
2. queries for parameter measurements
3. a library of primitive measurement functions

Object models

Our planner is provided with models of how an image is generated given the parameters of an object. For this purpose we use context-free attribute grammars. A program to measure parameters on an object performs the inverse of the grammatical generation function: it produces a parsing in the sense of the grammar. For our example the grammar can be expressed in the following substitution rule:

terminals: **d1(R,S), d2(R,S) = disk(R,S)**

twodisks(R,D,S) → [d1(R,S) , d2(R+D,S)]

This expresses that the two "parts" "d1" and "d2" are substituted for the "twodisks" object, and it gives the relations between the "attributes" R, S of dl and d2 and the attributes of the "twodisks" object.

We want the grammar has to meet several requirements:

the grammar should be, as much as possible, independent of the application (no details should be added per application)

the planner must be able to use the grammar in such a way that the complexity of the program depends on the application but not the complexity of the grammar.

The first requirement is only put to the test if we try the system on many kinds of application and not on just "a few" example images. We try to realise these requirements by including descriptions of objects at several levels of detail in a single grammar. For example, suppose we replace the "twodisks" by "twosquares" with component squares with slightly rounded corners; for most purposes a grammatical description as a perfect square may be sufficient, but for some applications the corner rounding must be included. If this is the case, we include both the perfect square description and the rounded square as alternatives in the grammar, not necessarily stating the relation between the two descriptions (in this case that the one is a more precise version of the other).

The price we have to pay for this is that in general the grammars are not entirely consistent. The idea is that this is not fatal because the resulting programs will be tested before use.

Queries for parameter measurements

The planner must be told what its goal measurement is. This will be done by an external agent. We feel that if our system is to be useful for current programmers we should not force them to give away all of the responsibility for programming at once. We realise this in our system by formulating the goal in terms of a program. This program may contain variables which are not given a value, but which are given meaning as attributes of the grammar. For example

```
with "twodisks":
    print(D,±2);
```

This prompts the planner to produce a program that determines D with a precision plus or minus 2, S, not being mentioned, does not have to be measured. If the external agent wants to take some work from the planner, he could give the more detailed goal program

```
with "twodisks":
    D = d2.R - d1.R;
    print(D,±2);
```

Now the planner can restrict itself to devising programs to determine R of dl and d2.

The library

A third determining factor is the availability of primitive functions. The planner must be provided with a library of such primitives, together with specifications connecting them to grammatical symbols and their attributes. An example b the "diskprograml" primitive mentioned above. Its formal entry in the library could be of the following form:

diskprograml (image(disk (R,S)) produces **R ±1, S ±2**
code reference; . . . . .

This expresses that diskprograml is a parsing function, which performs the inverse of the generation implied by the terminal symbol "disk" in the generative grammar. It also expresses the accuracy of the function. Preferably, the library will have to contain a number of alternative primitives, which differ in effectiveness in different contexts, for any terminal of the grammar, and possibly also for a number of non-terminals.

In our work we make use of PAPS (Picture Acquisition and Processing System), a vision system developed within PHILIPS for applications in factory automation . One of the strongest properties of this system is its ability to do template matching at tv-image speeds. Most of our primitives use this facility; they generate coordinates of primitive features as positions where the template has matched. In "diskprograml" we have implemented the disk position measurement in this way.

The planner

Having discussed the inputs to the planner we are now in a position to discuss the planning process itself. The planner must plan a program which calculates a goal variable. This calculation can be achieved in a number of ways:

1. express the variable in a generative way in other variables by means of the grammar
2. call a primitive that can measure the variable from an image
3. solve for the variable by inverting one or more of the generative relations in the grammar
   (This involves an equation solving system)

The planner tries to apply any of these rules to its goal. If it does so, any rule can recursively generate new goals. The planner continues to apply the rules on these new goals until no unknown variables remain. In the process the planner produces a trace of procedures that must be invoked in the task program. An example:

starting from **print(D,±2);**

where D of "twodisks" must be determined. The planner applies rule 3, solving D from **d1.R=R and d2.R=R+D:**

```
D = d2.R - d1.R;
print(D,±2);
```

where R of dI and d2 (both disks), must be determined. Applying rule 2 twice, the planner generates the final program:

```
d1 = diskprogram1 ( )
d2 = diskprogram1 ( )
D  = d2.R - d1.R;
print(D,±2);
```

where no more variables are unknown. Note that neither the equality of the two disk sites nor the distinctness of the two disks is verified in this program. This neglect of verification is even more obvious if we query for S: the goal program: print**(S,±3)** leads by rule (3) with dI.S=S and rule (2) to

```
d1 = diskprogram1 ( )
S  = d1.S
print(S,±3);
```

The second disk isn't even inspected here (of course there is no need for inspection if only images of pairs of equal sized disks are shown: the need for verification depends on the kind of scene).

In most realistic problems the planning process will be considerably more complicated, hence we have to use some pruning mechanism. This may lead the planner to miss solutions. For our purposes this is not a serious problem because usually our problems have many possible solutions and we are more interested in discovering good ones than in discovering all.

Finally, we must account for the fourth factor influencing machine vision programs: the kind of scene. In this step we must compensate for inaccuracies of our models. For example, in the twodisks model we stated nothing about what happens if the two disks come close together. Yet the selection of templates for the "diskprogramI" primitive is likely to be affected by such a factor. Alternatively, if there are many objects in the scene which closely resemble disks, we might want a template that distinguishes between them and a disk. The planner must choose a template appropriate for the task by testing in some example images.

This is done as follows: a teacher provides the planner with a number of example images, together with sets of attributes for objects present in these images. These sets are used, with the generative grammar, to calculate the desired outcome of statements in the generated programs. By comparing these desired results with the measurements obtained from selectable primitives the planner verifies the relations and primitives it selects.

Conclusion

The work described in this paper is a continuation of the work reported by Persoon . His problem: recognition of rigid objects from images by means of a learned object description, is also pursued by Bolles and Faugeras . We have added to this work measurement of attributes other than position, orientation and scale, (e.g. both disksise S and distance vector D in the "twodisk" model; both orientation PHI, length L and width W variable in a rectangle model). This is a non-trivial step because the estimation of position, orientation and scale from position observations of features can be expressed as a linear estimation problem. This can be solved by accumulating observations from an image in a least-squares estimator For non-linear problems a structurally more complicated estimator is needed.

This leads us in the direction of more general model-driven image analysis systems of which ACRONYM is a prominent example (for a review of related systems see). Our approach differs from this work in that we use fairly rigid object models, with only a limited number of strictly parametrised deformation modes. Moreover, we include an off-line vision planning stage, using examples. Hence we avoid the problem of interpreting a kind of scene offered for the first time to the system. In this way we try to economise on model complexity.

References

1. F.L.Thissen "The PHILIPS modular picture acquisition and processing system - PAPS", In Proc. 5th International conference on RObot Vision and SEnsory Control, Amsterdam 1985 N.J.Zimmerman (Ed.)
2. E.J.Persoon "Learning Algorithms applied to the processing of industrial images", Pattern Recognition in Practice, E.S. Gelsema, L.W. Kanal (Eds.) North Holland Publishing Co., Amsterdam 1980.
3. R.C.Bolles, R.A. Cain, "Recognising and locating partially visible objects: the local feature focus method", The International Journal of Robotics Research, 1:3, (1982) p57.
4. N.Ayache, O.D.Faugeras, "HYPER: a new approach for the recognition and positioning of two-dimensional objects", IEEE transactions PAMI-8 1 p44.
5. R.A.Brooks "Symbolic reasoning among 3-d models and 2-d images", Artificial Intelligence, 17 (1981) p285.
6. T. O. Binford "Survey of model based image analysis systems", The International Journal of Robotics Research, 1:1 (1982) pl8.