CBCV: A CAD-Based Computer Vision System

Thomas C. Henderson, John Evans, Lane Grayston
Allen Sanderson, Leigh Stoller and Eliot Weitz

Department of Computer Science
University of Utah
Salt Lake City, Utah 84112

Abstract

The CBCV system has been developed in order to provide the capability of automatically synthesizing executable vision modules for various functions like object recognition, pose determination, quality inspection, etc. A wide range of tools exist for both 2D and 3D vision, including not only software capabilities for various vision algorithms, but also a high-level frame-based system for describing knowledge about applications and the techniques for solving particular problems[1].

1 Introduction

Computer Aided Design, or CAD as it is better known, has many advantages to offer in the design and development of manufactured items. Computer vision, on the other hand, has yet to make major inroads into the manufacturing domain. We believe that a closer tie between geometric models and computer vision will lead to greater application of computer vision techniques in industry and to a more efficient and effective manufacturing process.

In this paper, we examine the role of Computer Aided Geometric Design models in providing support for computer vision techniques. In particular, we examine the requirements placed on CAD systems to achieve useful vision functions, and we take a look at the nature of the representation differences between CAD and vision, and we describe AI techniques for synthesizing executable computer vision modules based on an analysis of the task requirements, hardware and software available, and on the geometric object under consideration.

We give detailed examples of both CAD and computer vision systems, as well as synthesis techniques for automatically deriving visual inspection, object recognition, and pose determination modules. We use the Alpha_1 CAGD system, developed by Rich Riesenfeld and Elaine Cohen and their colleagues at the University of Utah for all of our design and 3D data manipulation. Their system is a boundary representation, B-spline based modeler, and provides exceptional design and analysis capabilities.

For the computer vision system, we use the IKS (Image Kernel System). This developed out of an early set of vision tools acquired from Bill Havens at the University of British Columbia. The majority of the current IKS system, however, has been developed over the last few years at the University of Utah. The functions range from low-level image to image processing routines to 3D intrinsic characteristic functions for the analysis of 3D range data.

The knowledge-based component of the system described here has been developed on top of PCLS (Portable Common Lisp Standard) in FROBS (FRames and OBjectS). These systems were developed by Bob Kessler and the PASS (Portable Ai Sys-

[1]This work was supported in part by the National Science Foundation under Grant INT-8909596. This work was also supported in part by DARPA (N00014-88-K-0689). This paper is a revised version of[11, 12].

temS) group at the University of Utah.

2 Computer Aided Geometric Design

2.1 General Considerations

Computer vision has been an active research area for over 25 years. In the past, emphasis was on low level processing such as intensity and signal processing to perform edge detection. More recently, models of objects and knowledge of the working environment have provided the basis for driving vision systems. This is known as model-based vision. The pursuit of the fully automated assembly environment has fueled interest in model-based computer vision and object manipulation. This involves building a 3-D model of the object, matching the sensed environment with the known world and determining the position and orientation of the recognized objects. The goal is to provide a solution to the problem of visual recognition in a well-known domain.

In the automation environment, recognition schemes and representations have typically been constructed using ad hoc techniques. Although objects used in the assembly process are designed with a CAD system, generally there is no direct link from the CAD system to the robotic workcell. This means the recognition systems are constructed independently of the CAD model database. What is desired is a systematic approach for both the generation of representations and recognition strategies based on the CAD models. Such a system provides an integrated automation environment. The system is composed of several components: a CAD system, a milling system, a recognition system and a manipulation system. In this paper, the automatic generation of recognition strategies based on the CAGD model is studied. It has also been determined that the use of shape, inherent in CAGD models, can also be used to drive the recognition process. Others have been studying portions of this system. Recent work by Ho has focused on the generation of computer vision models directly from a CAGD model[1, 14].

The work described here investigates the use of geometric knowledge in constructing 2D recognition codes. These codes provide a robust mechanism for recognition and localization of two-dimensional objects (occluded as well as non-occluded) in typical manufacturing scenes.

One of the first researchers to study the automatic synthesis of general recognition strategies was Goad[7]. He was concerned with automatic programming for 3-D model based vision. His work generated a recognition scheme for matching edges based on a general sequential matching algorithm. His system didn't consider partial occlusion. However, this was a major contribution since it was one of the first attempts to automate the generation of recognition schemes.

Another influential project was the 3DPO system by Bolles and Horaud[3]. This work is the 3-D generalization of the Local Feature Focus method[2]. Their system annotates a CAD model producing what is called the extended CAD model. From this model, feature analysis is performed to determine unique features from which to base hypotheses. The focus feature in their system is the dihedral arc. When the recognition system finds a dihedral arc, it looks for nearby features which are used to discriminate between model arcs with similar attributes. From these, an object's pose is hypothesized and subsequently verified.

Recently, Ikeuchi has explored the use of interpretation trees for representation of recognition strategies[15]. His system uses the concept of visible faces to generate generic representative views, called aspects. From this set of aspects, an interpretation tree is formed which discriminates among the different aspects. His system uses a variety of object features such as: EGI, face inertia, adjacency information, face shape, and surface characteristics. Most of these features are based on planar faces. Others to work on model-based vision include[4, 6, 8].

Hansen and Henderson[10] have also proposed a CAD-based 3D recognition and localization scheme based on strategy trees. That system isn't dependent on a certain class of features but rather can be extended to include many classes of features. The system also performs automatic selection of features based on a set of constraints: feature filters. These features are used to form a strategy tree which provides a scheme for hypothesis formation, corroborating evidence gathering and object verification. The flexibility of this approach makes it significantly different from related work.

Our main goal of is the automatic synthesis of recognition system specifications for CAD-based 2D and 3D computer vision[9, 10, 17]. Given a CAD model of an object, a specific, tailor-made system to recognize and locate the object is synthesized.

To attain this goal, the following problems have been studied:

1. **Geometric Knowledge Representation**: The use of geometric data is central to a strong recognition paradigm. Weak methods can only be avoided when better information is available. The Alpha_1 B-spline model allows the modeling of freeform sculptured surfaces. To obtain the geometric features of interest for 3-D recognition, techniques for the transformation to a computer vision representation have been developed.

2. **Automatic Feature Selection**: The part to be recognized or manipulated must be examined for significant features which can be reliably detected and which constrain the object's pose as much as possible. Moreover, such a set of features must cover the object from any possible viewing angle. In solving the feature selection problem, a technique is available for synthesizing recognition systems. This produces much more efficient, robust, reliable and comprehensible systems.

3. **Recognition System Synthesis**: Once a robust, complete and consistent set of features has been selected, a recognition strategy is automatically generated. Such a strategy takes into account the strongest features and how their presence in a scene constrains the remaining search. The features and the corresponding detection algorithms are welded, as optimally as possible, into a search process for object identification and pose determination. The automatic synthesis of search strategies is a great step forward toward the goal of automated manufacturing. Generation of strategies is constrained, not only by the feature selection process but, by the actual task to be accomplished. Thus, strategies for a specific task might not be as strong when applied to a different task; strategies are task specific.

Alpha_1 is an experimental CAGD based solid modeler system incorporating sculptured surfaces[5]. It allows in a single system both high quality computer graphics and freeform surface representation and design. It uses a rational polynomial spline representation of arbitrary degree to represent the basic shapes of the models. The Alpha_1 modeler allows the user to design an object by giving a sequence of commands. These commands define the geometry of the object. At the same time, the result of each command can be viewed in a separate window. Although specific Alpha_1 commands are given here, we use them to describe a more general philosophy of design. The underlying motivation is to exploit wherever possible the geometric modeling system functions to provide the information required to support computer vision applications.

2.2 Front Suspension Shock Linkage

To illustrate our modeling philosophy, we will use the design of a front suspension shock linkage. This part was designed by Samuel H. Drake at the University of Utah, and was actually milled and used in a small off-road vehicle built for the SAE Mini-Baja student competition.

The design is specified in such a way so as to facilitate the machining and automatic inspection of the part. The following commands bring in the (Lisp) definitions of design features (e.g., pockets, holes, etc.) and set up reference axes.

```
{
load features; – Load feature definitions
setSrfNorms(T);

Xref := XAxis; – Define coordinate axes
Yref := YAxis;
Zref := ZAxis;
};
```

Now, a set of bounding lines can be defined; the *reverseObj* command is used to keep the normal pointing into the shape (i.e., each bounding line has an orientation).

```
{
UpLinkConstLine1 := lineVertical( -0.5 );
UpLinkConstLine2 := lineVertical( 2.5 );
UpLinkConstLine3 := lineHorizontal( 0.75 );
UpLinkConstLine4 := lineHorizontal( -0.75 );
UpLinkConstLine5 := reverseObj(
        lineHorizontal( 0.5 ));
UpLinkConstLine6 := lineHorizontal( -0.5 );
UpLinkConstLine7 := lineHorizontal( 0.44 );
UpLinkConstLine8 := reverseObj(
        lineHorizontal( -0.44 ));
UpLinkConstLine9 := reverseObj(
        lineVertical( 0.875 ));
UpLinkConstLine10 := lineVertical( 1.5 );
```

Similar commands (not shown) define the remaining points, arcs, and segments needed to specify the shape. Note that the shape is defined in the x-y plane. The object's profile can now be defined:

```
UpLinkProfile1 := profile( UpLinkConstPt3,
    UpLinkConstArc3, UpLinkConstArc4,
UpLinkConstPt5, UpLinkConstPt8,
UpLinkConstArc1, UpLinkConstArc2,
UpLinkConstPt9, UpLinkConstPt10,
UpLinkConstArc5, UpLinkConstArc6,
UpLinkConstPt14, UpLinkConstPt3);
```

Now we define the bounding z positions and extrude the shape in 3D:

```
UpLinkExPt1 := pt( 0.0, 0.0, 0.55 );
UpLinkExPt2 := pt( 0.0, 0.0, -0.55 );
UpLinkShape := extrude( UpLinkProfile1,
    UpLinkExPt1, UpLinkExPt2, T, T );
};
```

Thus, the basic shape is defined. Next, we add a hole:

```
UplinkFixHole2 := objTransform( hole( origin,
 0.3125, 0.15, 0.0, T ), tx( 1.325 ), tz( 0.075 ));
```

We can view the hole shape with the profile shape. However, to actually put the hole through the profile surface, we need to perform a Boolean subtraction. First, we store the two shapes (as defined by the variables UpLinkShape and UplinkFixHole2) to a file named *UpLinkTestOp.a1*.

```
dumpa1File( list( UpLinkShape, UpLinkFixHole2
    ), "UpLinkTestOp.a1" );
```

At the shell command level, we then perform the Boolean set operation which subtracts the hole from the main shape:

```
cs > set_op < UpLinkTestOp.a1 > UpOut.a1
```

The final object is shown in Figure 1.

The geometric specification facilitates the direct extraction of object features. However, many modeling systems do not allow this. Therefore, the easiest way to provide models for a 2D vision system is to render an image from the CAD model and provide that image as the training set to the 2D vision system.

Figure 1: Rendering of Linkage

3 The FROBS Knowledge and Rule Base

The knowledge-base component of CBCV is written in FROBS (FRames + OBjectS) which is an object-oriented frames package that runs on top of Common-Lisp and provides:

- object oriented programming
- frame based programming
- daemons
- rule based programming.

An overview of the system is given here; for more details, see the FROBS Manual[16].

3.1 Overview of FROBS

The basic building block of the FROBS package is called a module. Modules consist of a class FROB and all of its associated methods. This provides for total method and data access hiding with no distinction between methods and slots. The organization of class FROBs can be viewed as a tree structure, although more complicated schema-type structures are possible through multiple inheritance. FROB class instances are leaves of the tree.

The class frob is used to define the structure of instance frobs of that class. It is also the frob that **daemons** and **methods** are defined over. Inheritance of **methods** is done through the class frobs. A special feature of the class frob is that it is an instance of itself. It can be used like any other instance of the class. Figure 2 shows how an algorithm class FROB and a subclass FROB are defined.

FROBS are used to build both the knowledge-based vision system and the application system it synthesizes. This allows templates in the knowledge-based system to be directly used in the application system. The concept of logical sensors is implemented easily using objects to form logical sensors[13]. Class FROBS represent logical sensor templates to be instantiated for application system synthesis.

Most importantly, the FROBS package provides forward chaining rules as well as slot daemons. Slot daemons are useful for automatic data consistency checking and hidden slot calculations. The forward chaining rules provide the mechanism needed to create the knowledge base.

3.1.1 System Support

The knowledge-based system must have utilities for supporting the networking of logical sensors and objects. These utilities provide the foundation from which the system is built. Higher level utilities are built on top of lower level ones for sophisticated system operations. The lowest level utility functions should have a maximal amount of flexibility since it is not known what or how more powerful constructs built upon them will be used. In the prototype system they are implemented as methods attached to FROB classes which define major components of the system. These classes and their methods form the templates from which application systems are synthesized. The application specific rules use knowledge of these templates to apply the line interpretation rules by relying on the transparent nature of the methods to handle lower level hardware or operating system specific tasks.

An example is the FROB representing the class of cameras. Knowledge about operating this class of

```
(def-class algorithm nil
        :slots (name
                size
                language
                machine))

(def-class feature-calculator ({class algorithm})
        :slots (feature-type
                focus-type))
```

Figure 2: Example of FROB Class Definition

```
(def-rule select-lff
        :type ((?req requirements))
        :prem((not (member 'lff (applications
                ?req)))
                (equal 'recognition (task
                ?req)))
        :conc ((assert-val ?req 'applications
                (cons 'lff (applications
                ?req)))
        (make class system-specs
                :task 'recognizer
                :method 'lff
                :time (time ?req)
                :space (space ?req)
                :accuracy (accuracy
                ?req))))
```

Figure 3: A FROB Forward Chaining Rule

camera is represented in a "run" method which is local to the class. It executes operating system commands which are not of concern to the object using the method. A "run" method is also provided to other sensors which have other operating system commands which are transparent to the caller of the method.

3.1.2 Language Issues

Since the application system is created from FROBS in the same environment as the knowledge-based system, the application system runs in the Common Lisp environment. To require that all of the algorithms in the system be written in Common Lisp would be a severe restriction to its flexibility. The object-based approach allows algorithms written in any language to be incorporated into the system as an algorithm object.

Methods are used to run the algorithm and provide it with the necessary I/O. Since the internal representation of the object is transparent to I/O from the outside, algorithms written in any language can be incorporated into the system as long as there are low level utilities in the system to support the methods which run them.

4 2D Capabilities Applications

Although the general goal of CAD-Based Computer Vision is to recover the 3D nature of the objects in

a scene, many tasks can be handled using 2D techniques. This amounts to extracting regions of interest from the 2D image and analyzing the features of those regions. There are many approaches to this problem, and we present two simple techniques which are available in CBCV: global feature matching and Local Feature Focus.

Both techniques require that objects in the image be separated from the background and that distinct connected regions have unique labels. Then, features are computed for each region. Global features are derived from some measure of the entire set of pixels of the region; for example, area is a global feature. Local features are those which are restricted in spatial extent and only require a small percentage of pixels from the region; corners and holes are examples of local features.

Figure 4 shows the steps involved in applying these matching techniques. The training data is used to construct a model of the object under consideration. As indicated earlier, such a model is usually based on visual features of the object; other kinds of features could be used, such as weight or surface roughness, but those will not be considered here. The standard approach to get the visual features of the object is to examine several views (digital images) of the object. These then constitute the training data, and after the features of the object are extracted, then some statistical analysis of the features is performed. Robust features are then selected to represent the object; that is, the mean values of the features are determined as well as their variances. Finally, some sort of comparison measure is selected; this includes both distance functions (e.g., Euclidean, Mahalanobis, Manhattan), as well as similarity measures (e.g., the correlation coefficient).

Figure 4: Vision Paradigm

The images used as training data can be obtained several ways. Sometimes an actual part is available and images are taken from the camera and image acquisition system. When a CAD model is available, the test images can be produced by rendering several views of the object. Alternatively, one view is sufficient if the statistical properties of the feature calculation processes are known.

It may be possible to determine the vision model directly from the CAD model without resorting to the use of images. For example, the surface area of a face can be calculated from the definition of the profile curve. When using manufacturing features such as pockets or holes, their dimensions are usually part of the definition of the part. In this case, ideal values are obtained, and it is necessary to take into account the error introduced by the manufacturing process and the image acquisition system.

4.1 Global Feature Matching

The *feature class* is defined as a very simple class consisting of these slots:

- name: the name of the feature,
- command: the executable command line, and
- features: a list comprised of:
  - a select switch (T or NIL)
  - the mean value of the feature, and
  - the variance of the feature.

The **FROB** definition for this is:

```
(def-class feature nil
:slots (name
        command
        features
))
```

Once the *feature class* has been defined, individual features can be defined. For example, the *area feature* is defined as:

```
(def-class area class feature
:init ((nil (name 'area)
            (command "path/area")))
)
```

Thus, *area* is a *feature* and has the name 'area' and is invoked by running '*path*/area' at the shell level. Of course, '*path*' must be expanded into the correct path to the area binary file.

Other features are defined similarly: *aspect, diameter, n1, n2, n3, n4, n5, n6, n7, perimeter* and *thinness*. These frames comprise the knowledge (at this level of the *ISA* hierarchy) about features. However, to create executable instances requires the definition of the 'make' method. for example:

```
(def-method (class area make) ()
   (let ((instance (new-instance $self)))
     instance))
```

defines a method for making instances of the *area* feature.

Now we can define useful methods which operate on features. The most basic operation is to run the feature operator on an image to produce a feature value for each connected region in the image:

```
(def-method (class feature run) (inputdata)
  (let ((stream (start-feature-process inputdata
          (command $self))))
    (unwind-protect
      (read-feature-output stream)
      (close stream))))
```

The *start-feature-process* function takes inputdata as the segmented image and passes it to the feature command line. The output is then piped into the variable *stream* and is read from there by the *read-feature-output* function which is returned as the value of the method.

For example, to run the area feature on the image *scene.img*, the following command is issued:

```
(run {class area} "scene.img")
```

4.1.1 An Example

We now give an example of CBCV using the global feature matching technique to inspect the linkage.

The training command is given first:

```
(setq *r* (make-recognizer :hint 'global
   :training-data *crocfiles*))
```

where *crocfiles* is a FROB variable that has the name of the training files. The result is an algorithm:

Figure 5: Scene Image for Global Matching

```
ALGORITHM 0 has the following values:
(ALGORITHM ALGORITHM) ⇒ GLOBAL 0
(ALGORITHM HINT) ⇒ GLOBAL
(ALGORITHM OCCLUSION) ⇒ NIL
(ALGORITHM SPACE) ⇒ NIL
(ALGORITHM TIME) ⇒ NIL
(ALGORITHM NAME) ⇒ NIL
```

All features are tested for robustness, and in this case, only perimeter, n7, diameter, and area survive. Next, the global matcher can be run on a scene image (see Figure 5):

```
(match *r* :image-data "scene-image.img"))
```

Then the results of the global analysis are reported:

```
Feature match success: "scene1-seg.img"
T
```

4.2 Local Feature Matching

The Local Feature Focus method proposed by Bolles[2] is a robust 2D shape recognition and localization scheme. The method is organized as follows:

- Model Building
  1. Enumeration of potentially useful features
  2. Location of structurally equivalent features
  3. Enumeration of secondary feature groups
  4. Selection of secondary feature groups
  5. Ranking of focus features
- Recognition and Localization
  1. Get scene features

Figure 6: Model Superimposed on Scene

2. Match focus graph
3. Generate hypothesis (including pose transformation)
4. Verify match

4.2.1 An Example

We will now use this technique to locate the linkage part. The CBCV system invokes a sequence of filters to obtain the features from the CAD image of the object. First, a boundary file is produced. Next, the center of mass of the object is found: (104.326981 104.861526); this is used to produce feature descriptions located with respect to the object's center of mass. The corners are found next. From this information, the feature types are determined. From this information, the focus features are determined. This then constitutes the model.

Given a scene, as shown in Figure 5, the system must discover the transformation. The features in the scene are determined in much the same way as for the model image. After matching model features to scene features, the system identifies the transformation, and produces the match shown in Figure 6.

5 Current Work

We are currently working on integrating the 3D strategy tree approach into the knowledge-based system (CBCV) described above. We hope to be able to model freeform surfaces of objects in the Alpha_1 CAGD system, and to automatically synthesize recognition or pose recovery executable codes by analysis of the features. Note that manufacturing features such as holes, pockets, etc. are a direct part of the Alpha_1 design language. Once the discovery of complete, consistent and robust feature sets from the CAGD model can be done reliably, then range data analysis routines can be invoked to solve the particular application problem.

These methods can also be used for reverse engineering, as well as for object validation. We have recently acquired a very precise coordinate measurement machine; a coarse to fine approach is under development for producing a CAGD model directly from the measurements of the object.

The ultimate goals include the ability to:

- **validate** any object for which a CAGD model exists, and
- **reverse engineer** a CAGD model from a physical instance of an object.

Acknowledgements

We would like to thank the members of the Alpha_1 group for putting the CAD into our CAD-Based vision work.

References

[1] B. Bhanu and C.C. Ho. CAGD-Based 3-D Object Representations for Computer Vision. *IEEE Computer*, 20(8):19–36, August 1987.

[2] R.C. Bolles and R.A. Cain. Recognizing and Locating Partially Visible Objects: The Local-Feature-Focus Method. *Robotics Research*, 1(3):57–82, 1982.

[3] R.C. Bolles and P. Horaud. 3DPO: A Three-Dimensional Part Orientation System. *Robotics Research*, 5(3):3–26, 1986.

[4] C.H. Chen and A.C. Kak. A Robot Vision System for Recognizing 3-D Objects in Low-Order Polynomial Time. *IEEE Transactions on Systems, Man and Cybernetics*, 19(6):1535–1563, 1989.

[5] E. Cohen. Some Mathematical Tools for a Modeler's Workbench. *IEEE Computer Graphics and Applications*, 63–66, October 1983.

[6] P.J. Flynn and A.K. Jain. CAD-Based Computer Vision: From CAD Models to Relational Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(2):114–132, 1991.

[7] C. Goad. Special Purpose, Automatic Programming for 3D Model-Based Vision. In *Proceedings of the DARPA Image Understanding Workshop*, pages 94–104, DARPA, 1983.

[8] E. Grimson and T. Lozano-Perez. Model-Based Recognition and Localization from Sparse Range or Tactile Data. *Robotics Research*, 3(3):3–35, Fall 1984.

[9] C.D. Hansen. *CAGD-Based Computer Vision: The Automatic Generation of Recognition Strategies.* PhD thesis, The University of Utah, Salt Lake City, Utah, July 1988.

[10] Charles D. Hansen and Thomas C. Henderson. CAGD-Based Computer Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-11(10):1181–1193, 1989.

[11] Thomas C. Henderson, J. Evans, L. Grayston, A. Sanderson, L. Stoller, and E. Weitz. *CBCV: A CAD-Based Vision System.* Technical Report UU-CS-90-013, University of Utah, Department of Computer Science, June 1990.

[12] Thomas C. Henderson, J. Evans, L. Grayston, A. Sanderson, L. Stoller, and E. Weitz. CBCV: A CAD-Based Vision System. *Bild und Ton*, 43(12):364–367, 1990.

[13] Thomas C. Henderson, Eliot Weitz, Chuck Hansen, and Amar Mitiche. Multisensor Knowledge Systems: Interpreting 3D Structure. *International Journal of Robotics Research*, 7(6):114–137, 1988.

[14] C.C. Ho. *CAGD-Based 3-D Object Representations for Computer Vision.* Master's thesis, University of Utah, Salt Lake City, Utah, June 1987.

[15] K. Ikeuchi. Model-Based Interpretation of Range Imagery. In *Proceedings of the DARPA Image Understanding Workshop*, pages 321–339, DARPA, 1987.

[16] Eric Muehle. *FROBS Manual.* Technical Report PASS-note-86-11, University of Utah, October 1986.

[17] Eliot Weitz. *Knowledge-Based 2D Vision System Synthesis.* Master's thesis, University of Utah, Salt Lake City, Utah, June 1987.