18 Automated Visual Assembly Inspection

KHALID W. KHAWAJA

Structural Dynamics Research Corporation, Milford, Ohio 45150

DANIEL TRETTER

Hewlett-Packard Company, Palo Alto, California 94304-1126

ANTHONY A. MACIEJEWSKI
CHARLES A. BOUMAN

Computer and Electrical Engineering Department, Purdue University, West Lafayette, Indiana 47907-1285

I. INTRODUCTION

At a time when quality and cost are becoming even more important in the manufacturing process, accurate and efficient inspection is critical. However, the complexity of electrical and mechanical assemblies has reached a point where human inspection can be fatiguing, unreliable, and expensive. This has prompted many manufacturers to implement automated visual inspection systems. Unfortunately, efforts to achieve the advantages of CAD-driven automatic sensor planning and visual inspection systems for three-dimensional assemblies have been largely unrealized.

The automated inspection task includes two interrelated components: the setup and configuration of the inspection cell, including camera and light placement,

Expert Systems, Edited by Cornelius T. Leondes, Volume 3

ISBN: 0-12-443883-0 / $35.00

and the algorithm used to analyze the captured image and to determine whether assembly errors have occurred. Both of these components can benefit from the use of intelligent and adaptive systems approaches. In this chapter we design a general model and structural form for each component and allow the CAD model of a given assembly to determine the appropriate model parameters and exact structure to be used when the assembly is inspected. This CAD-driven approach allows our system to adapt to a wide variety of assemblies automatically with little or no user interaction.

Sensor planning is used to configure the layout of the inspection cell for the efficient detection of assembly errors. Sensor planning for computer vision tasks has received some attention in recent years. One specific area that has been emphasized and is closest to this work is that of sensor planning for object feature detection [1]. For example, in [2] the region of viewpoints that satisfy a set of constraints is calculated. These constraints are formulated in terms of resolution, focus, field of view, and visibility requirements for a set of object features like points, lines, and faces. A function is then formulated that attempts to find an optimal viewpoint in this region. In [3] constraints are placed on both a camera and a point light source location for observing object polygons. Each constraint results in a region of admissible points. For light regions, locations associated with the specular direction of the inspected faces are excluded. The intersection of these regions yields the final admissible camera and light location points. Moreover, in [4] the problem of automatic sensor and light positioning is considered in terms of optimizing a function that describes edge visibility, where both the camera and light locations are constrained to lie on a spherical surface with the center of interest lying at the sphere center. Similarly, in [5] an object point of interest is surrounded by a tessellated sphere on which a camera location is to be determined such that the point is not occluded. The points that maximize a formulated distance requirement are selected. Later, the work was extended to planning light source placement for recognizing objects with Lambertian surfaces [6]. In [7] an illumination planner for convex Lambertian objects is discussed, where reliable regions around the object are identified for the placement of several light sources.

Although object inspection is a possible application of these cited works, the extension of these approaches to assembly inspection has not been considered. The mere repetition of the inspection algorithms for each component of an assembly would be computationally unreasonable. Ideally, sensor planning for assembly inspection should be done to optimize the performance of the inspection algorithm being used. In this work we develop new algorithms for automatic camera and light source placement with this aim in mind. Our algorithms use the CAD information built into an assembly model along with specialized computer graphics hardware to accomplish this task. Fast rendering techniques are used to acquire needed data for the camera and light placement automation process. Moreover, realistic synthetic images are generated and used to run simulations for design and test purposes. To automate the camera placement, the CAD information for components of interest is used to analytically constrain the camera to a small region of the solution space. A function that measures the quality of a camera view is created. Then, once the camera location is known, an algorithm is designed to place a point light source. A second function is created to evaluate the quality of the light location. The camera and light quality functions are used in conjunction to select an optimal camera–light pair following a generate-and-test approach in the constrained solution space. One of the main advantages of this approach is that it uses the fact that an assembly, versus a single component, is being inspected. Currently, the approach assumes the components are assembled only with the use of vertical insertion operations.

In addition to the algorithms for determining camera and light placement, we develop an automated inspection algorithm to detect assembly errors from a single monochrome image of the object. The algorithm uses a novel multiscale stochastic image model to describe the appearance of a complex three-dimensional object in a two-dimensional monochrome image. This formal image model is used in conjunction with Bayesian estimation techniques to perform automated inspection. The model is based on a stochastic tree structure in which each node is an important subassembly of the three-dimensional object. The data associated with each node or subassembly are modeled in a wavelet domain. We use a fast multiscale search technique to compute the sequential maximum a posteriori (SMAP) estimate of the unknown position, scale factor, and two-dimensional rotation for each subassembly. The search is carried out in a manner similar to that of a sequential likelihood ratio test, where the process advances in scale rather than time. The results of this search determine whether the object passes inspection. A similar search is used in conjunction with the expectation maximization (EM) algorithm to estimate the model parameters for a given object from realistic training images generated synthetically from the CAD model of the assembly.

Low-level image models describe the behavior of individual image pixels relative to one another. Markov random fields and other spatial interaction models have proved useful for a variety of applications, including image segmentation and restoration [8, 9]. Bouman and Shapiro [10], along with Willsky, Benveniste, and their associates [11, 12], have developed multiscale stochastic models for image data. High-level image models are generally used to describe a more restrictive class of images. These models describe larger structures in the image explicitly, rather than describing individual pixel interactions. Grenander and his associates, for example, propose a model based on deformable templates to describe images of nonrigid objects [13], whereas Kopec and his colleagues model document images with the use of a Markov source model for symbol generation in conjunction with a noisy channel [14, 15]. Our image model is primarily high level, although we do model individual pixel statistics within the context of larger structures. In addition, we combine the image model with a fast multiscale search procedure to form an object detection algorithm for use in the particular application of automated inspection. Because the detection process is based on a formal model of the image data, it can be carried out in a consistent manner with the use of well-known stochastic estimation techniques.

A number of different approaches to the object inspection problem have been taken in the past. Much of the early work in this area concentrated on special-purpose algorithms for the inspection of specific objects [16]. More recently, inspection has often been viewed as only one of a number of related machine vision tasks, so general object recognition systems are used for inspection. Examples of this approach include Brooks' ACRONYM system [17], as well as the systems of Flynn and Jain [18] and Mehrotra and Grosky [19], which perform three-dimensional pose estimation and use a multiple-object database. Most object recognition techniques, however, are not based on a formal probabilistic model of the data. Instead, they generally extract features of some sort from the data and match these to corresponding object characteristics. Because our algorithm and image model were constructed with assembly inspection specifically in mind, we can take advantage of some of the unique characteristics of such a system, such as the highly controlled viewing environment and well-defined goal of the system. For this constrained problem, it is possible to develop an explicit stochastic model that can be used to guide the design of our algorithm.

Our model uses a stochastic tree structure in which each node is an important subassembly of the three-dimensional object. The important subassemblies and linkages between the nodes of the tree are automatically identified from the CAD information. Thus, the stochastic tree used for a given assembly will automatically adjust to best take advantage of that object's structure and the common assembly errors associated with its construction.

As illustrated in Fig. 1, our system components work together synergistically to form a complete inspection system. By using a well-defined problem structure (visual inspection of rigid assemblies from a single monochrome image) and

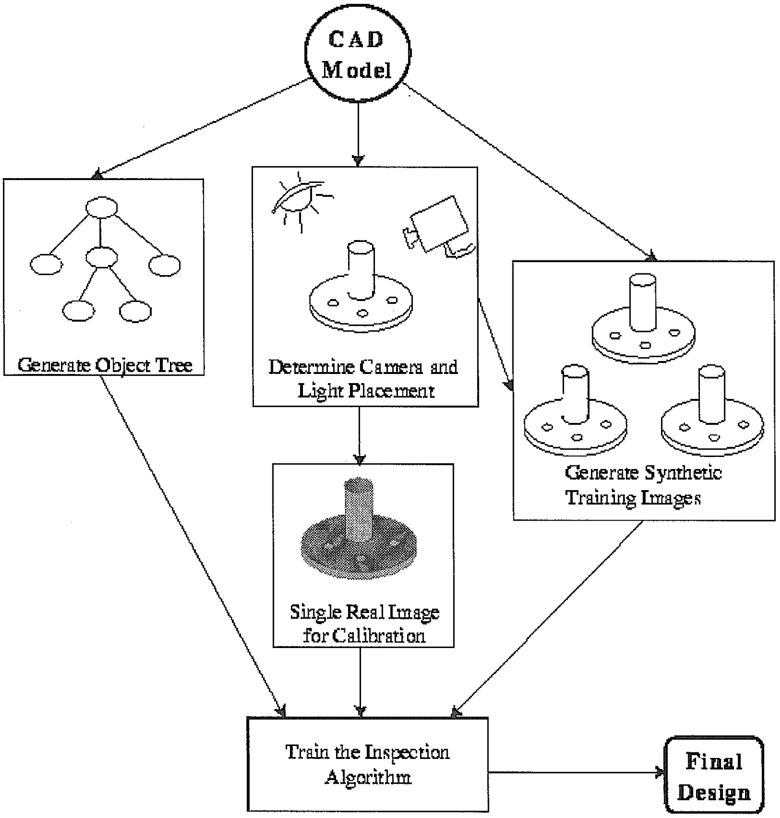

CAD Model
Generate Object Tree
Determine Camera and Light Placement
Generate Synthetic Training Images
Single Real Image for Calibration
Train the Inspection Algorithm
Final Design

FIGURE 1 The CAD model is used to automatically generate our assembly model and inspection conditions. The information from the model is used to train the inspection algorithm and adapt it to the assembly of interest.

assuming the availability of considerable information about the data (as embodied in CAD models of the assembly components and materials), our system can automatically adapt to a wide variety of objects with little or no user interaction. Although the computation required to specify and train the system for a specific assembly can be considerable, it is largely automatic in nature and only needs to be performed once for a given assembly. Thus, it can be done offline in parallel with other manufacturing tasks.

II. THE INSPECTION ALGORITHM

We approach automated inspection as a problem in object detection, where it is assumed that the inspection algorithm must make decisions based on a monochrome image of the object. In this section, we develop a model-based inspection algorithm designed to detect assembly errors in a rigid object from a single monochrome image of the object. Because the algorithm is designed specifically for automated inspection, we can take advantage of the highly structured viewing conditions typically found in a factory environment. For example, because the object to be inspected is known in advance, the algorithm is trained to be sensitive only to this one object; anything else in the field of view is taken to be extraneous to the inspection task. Furthermore, the regions of the object in which assembly errors are most likely to be visible are known (derived from the CAD model at an earlier stage of processing), so the algorithm concentrates most of its attention on those object regions. Finally, the approximate location and pose of the inspected part will often be known [20, 21]. The algorithm is therefore designed to be robust to limited changes in viewing conditions, but it does not allow for arbitrary object orientation. This algorithm is described in detail in [22].

The apparent shape and appearance of an object will alter because of slight changes in viewing conditions and allowed variations in component sizes and assembly construction, so the object model must be flexible enough to allow some degree of distortion. Each of the important features, or subassemblies, of the object is therefore modeled separately, and their relative positions in the image are permitted to vary randomly to a certain degree. The subassemblies are linked together in a stochastic tree structure, where the position, or state, of each subassembly is taken to be a random quantity dependent on the state of the parent subassembly in the tree. The states thus form a Bayesian network on the object tree [23].

Each subassembly is modeled separately with the use of the structure shown in Fig. 2, where the arrows indicate conditional dependence. A subassembly's location, scale, and orientation in the image are expressed as a random state vector **X**, where the component distributions are determined by the allowed viewing conditions. The exact distribution of **X** is dependent on the deterministic parameter

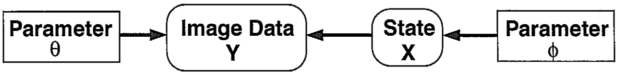


FIGURE 2 General model structure for a subassembly. The state is the (random) location, orientation, and scale factor of the subassembly. The image data are the (random) wavelet transform image. The parameters are deterministic quantities estimated from training data.

set ϕ, which will remain the same for all images. The parameters are estimated from a set of training images generated from CAD information, allowing the model to adapt to specific viewing conditions.

The data associated with each subassembly, which is taken to be a multiresolution wavelet decomposition of the original grayscale image, is modeled as a multiscale random field. Data values depend on the deterministic parameter θ, which can be thought of as a multiresolution template describing the appearance of the subassembly. The multiscale data model was developed with concepts and results from the theory of multiscale random processes in mind [10–12].

The inspection algorithm locates an object and all of its subassemblies in an image by estimating the state of each node of the object tree. The states are estimated based on the image data, which is modeled as a set of noisy measurements dependent on the underlying states. Thus, because the states form a Bayesian network on the object tree, the state estimation procedure is exactly analogous to state estimation for a hidden Markov model. The state estimation takes the form of a multiscale search at each node, progressing from the root of the object tree to its leaves. Each subassembly is inspected in turn, and the estimated state of the parent node is used to guide the multiscale search. The search at each node results in an approximation to the maximum posteriori state estimate for the associated subassembly, given the estimated parent state and the image data. The estimation procedure is therefore the SMAP procedure of Bouman and Shapiro [10]. This gives a noniterative, computationally efficient formulation for locating and identifying the desired object.

A similar multiscale search procedure is used during the training phase of the algorithm, where we estimate the model parameters from a set of training images. The parameter estimates are computed with the use of the iterative EM algorithm [24].

This inspection algorithm interacts with the rest of the inspection system primarily through two different mechanisms. The system uses the CAD model of the assembly to guide the construction of the object tree, with important subassemblies and linkages among them being identified and constructed automatically. The CAD model is also used to generate a variety of training images, each of which represents an "in-spec" assembly. The training images, in turn, are used to estimate model parameters. Thus, both the model structure and the model parameters adapt to the assembly through the CAD model.

This section is organized as follows. In Section II.A we define the tree structure making up the object model and specify the model associated with each subassembly. This model is then used in Section II.B to develop the multiscale search procedure for state estimation. Finally, Section II.C discusses our parameter estimation procedure, which is used to adapt the algorithm to the particular object of interest.

A. The Model

In this subsection, we will specify a formal stochastic image model that can be used to describe the appearance of a general class of complex three-dimensional objects. The model has two distinct levels to its structure: the object tree and the subassembly. Each node of the object tree will be used to represent the relative position and orientation of the important object features, called subassemblies.

Each subassembly will then be modeled with the use of a wavelet transform of the associated image region.

Figure 3 shows an example of an object tree for a complex three-dimensional object. Each box represents a subassembly or node of the tree and is drawn around a feature of interest in the object's image. The boxes are connected by lines into a tree structure, and the level of each node in the tree is represented by the number of lines making up the box. In general, the subassemblies will consist of various object components important for locating the object and for detecting assembly errors. Typically, nodes near the root of the tree are associated with larger parts of the object and represent the object's gross structure. These nodes also prove useful in locating the object in an image. Nodes further down the tree "zoom in" on smaller features that contain significant fine detail. The object tree is constructed from the CAD model of the assembly, which can be used to predict which features will be of the most use in detecting assembly errors.

Figure 4 illustrates the structure and conditional dependencies in an object tree. Each node is represented by an oval containing four quantities, $X^{(c)}$, $\phi^{(c)}$, Y, and $\theta^{(c)}$, where c is the index of the node, and arrows indicate conditional dependency. We will use uppercase letters to denote random quantities and lowercase letters for nonrandom sample realizations.

The random state $X^{(c)}$ contains the position, orientation, and scale of the subassembly. $X^{(c)}$ is assumed to be random because the geometry of the camera and object may vary from image to image. In general, however, the position of a subassembly will depend on the position of its parent node in the object tree. This conditional dependence is indicated by the arrows between nodes. Because the observed image depends on the location and orientation of the object and its components, the image data Y in Fig. 4 depend on each of the states, $X^{(c)}$.

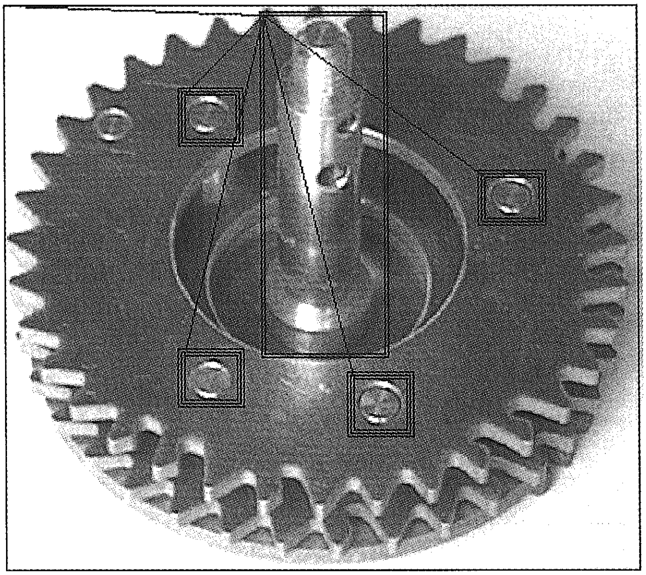


FIGURE 3 An initialization image is used to define the object tree. The boxes indicate the subassemblies associated with the nodes of the tree, and the lines connecting the boxes show the parent–child links.

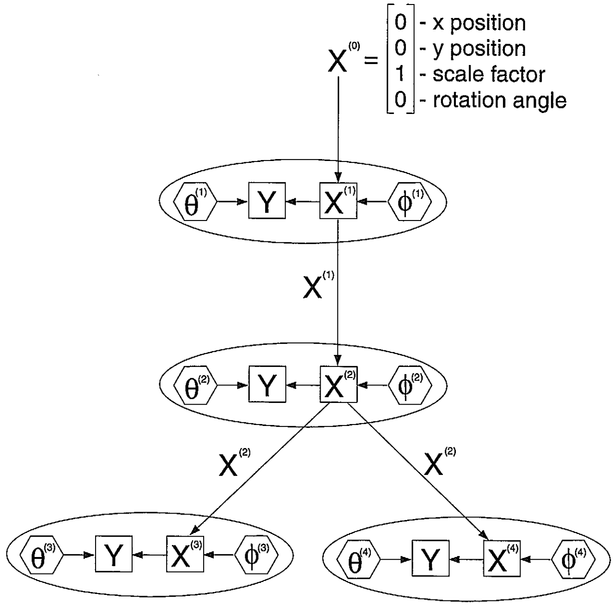


FIGURE 4 Model structure of complete object assembly. At each node, Y is the image data; $X^{(c)}$ is the state containing the position, orientation, and scale of the subassembly; $\theta^{(c)}$ is a set of data parameters that describes the appearance of the subassembly; and $\phi^{(c)}$ is a state parameter vector describing the variation in subassembly position.

In addition to random quantities, each node contains two deterministic parameter vectors, $\phi^{(c)}$ and $\theta^{(c)}$. These parameter vectors are used to adapt the model to a wide variety of possible object behaviors and imaging environments. The parameter $\phi^{(c)}$ determines the mean and variation of a node's state given the parent node's state, and $\theta^{(c)}$ determines the mean and variation of image pixels given the node's state. Intuitively, one might think of $\theta^{(c)}$ as containing an image template for the subassembly, but we will see that $\theta^{(c)}$ actually contains more information than a simple template.

Because subassemblies only depend on each other through their positions, the node states $X^{(c)}$ form a Markov chain along any path from the root to a leaf of the tree. This tree-dependent structure captures the interdependencies among the subassemblies while remaining amenable to efficient computational schemes [10, 12, 25]. If we index the nodes from 1 to M, then this Markov relationship may be stated as

$$p\left(x^{(1)}, \ldots, x^{(M)} \mid \phi^{(1)}, \ldots, \phi^{(M)}\right) = \prod_{c=1}^{M} p\left(x^{(c)} \mid X^{(p)} = x^{(p)}, \phi^{(c)}\right), \qquad (1)$$

where p denotes the parent of node c, and the parent state for the root node of the object tree is the deterministic state vector $x^{(0)}$. Notice that the state of the subassembly $X^{(c)}$ depends on both the state parameters $\phi^{(c)}$ and the state of the parent node $X^{(p)}$.

The density functions given in (1) must next be defined. The subassembly state has components $X^{(c)} = [S^t, Z, R]^t$, where $S = [S_{\mathrm{v}}, S_{\mathrm{h}}]^t$ is vertical and horizontal position, Z is the scale factor, and R is the angle of rotation in radians. The state $X^{(c)} = x^{(c)} = [(s^{(c)})^t,\ z^{(c)},\ r^{(c)}]^t$ defines a transformation of the subassembly from the image coordinate system to a normalized coordinate system with scale factor 1 and rotation angle 0. This normalized coordinate system is essentially used for data registration; the distortions in a particular image are undone, and the subassembly data are mapped to a common location. Each image pixel location i at resolution l will transform to a normalized location i', where

$$i' = \mathbf{T}^{(c)}(i - 2^{-l}s^{(c)}),$$
$$\mathbf{T}^{(c)} = \frac{1}{z^{(c)}}\begin{bmatrix} \cos r^{(c)} & \sin r^{(c)} \\ -\sin r^{(c)} & \cos r^{(c)} \end{bmatrix}.$$

We will use the matrix $\mathbf{T}^{(c)}$ to simplify our model notation.

The state parameter vector $\phi^{(c)}$ has the components

$$\phi^{(c)} = \begin{bmatrix} m^{(c)} \\ \gamma^{(c)} \end{bmatrix} = \left[(m_s^{(c)})^t, m_z^{(c)}, m_r^{(c)}, \gamma_s^{(c)}, \gamma_z^{(c)}, \gamma_r^{(c)}\right]^t,$$

where $m^{(c)}$ and $\gamma^{(c)}$ play the roles of mean and variance vectors, respectively. Given this notation, the state vector has a Gaussian distribution with the form

$$p\left(x^{(c)} \mid X^{(p)} = x^{(p)}, \phi^{(c)}\right) = \frac{|\mathbf{B}|^{-1/2}}{(2\pi)^2}\exp\left\{-\frac{1}{2}\left(x^{(c)} - x^{(p)} - \mathbf{A}m^{(c)}\right)^t \mathbf{B}^{-1}\left(x^{(c)} - x^{(p)} - \mathbf{A}m^{(c)}\right)\right\}, \quad (2)$$

where $\mathbf{A}$ is a matrix determined by the parent state $x^{(p)}$ through the transformation $\mathbf{T}^{(p)}$, and $\mathbf{B}$ is a matrix determined by $\phi^{(c)}$ and $x^{(p)}$:

$$\mathbf{A} = \begin{bmatrix} (\mathbf{T}^{(p)})^{-1} & \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix} \\ \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix} & \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} \left(z^{(p)}\right)^2\gamma_s^{(c)} & 0 & 0 & 0 \\ 0 & \left(z^{(p)}\right)^2\gamma_s^{(c)} & 0 & 0 \\ 0 & 0 & \gamma_z^{(c)} & 0 \\ 0 & 0 & 0 & \gamma_r^{(c)} \end{bmatrix}.$$

Note that the vertical and horizontal offset means depend on the matrix $\mathbf{T}^{(p)}$, which is a function of the scale factor $z^{(p)}$ and the rotation $r^{(p)}$. Therefore, the vertical and horizontal distances between subassemblies will scale with object size and change as the assembly rotates. For simplicity we assume that the vertical and horizontal positions have the same variance. This assumption makes the variances independent of rotation angle. The root node does not have an actual parent node, so for this node we define the parent state $X^{(0)}$ to be $x^{(0)} = [0, 0, 1, 0]^t$.

Having defined the relationship between the nodes of the object tree, we now need to construct a model for each subassembly or node of the tree. This model determines the distribution of the image pixels in the region of each subassembly. The subassembly model is based on a wavelet transform of the image. The wavelet transform has two important advantages in modeling of the image. First, because the transform may be thought of as approximately separating the image into distinct spatial frequency bands, it tends to decorrelate the image data [26]. We will

$$a)\ \begin{array}{cc} +1 & +1 \\ +1 & +1 \end{array} \qquad b)\ \begin{array}{cc} +1 & -1 \\ +1 & -1 \end{array}$$

$$c)\ \begin{array}{cc} +1 & +1 \\ -1 & -1 \end{array} \qquad d)\ \begin{array}{cc} -1 & +1 \\ +1 & -1 \end{array}$$

FIGURE 5 Basis functions for the Haar transform. Notice that *a* is the average, *b* is the vertical edge gradient, *c* is the horizontal edge gradient, and *d* is only responsive to thin diagonal lines.

see that this decorrelation removes undesirable mismatches caused by small shifts in the average gray scale. The decorrelation also results in a transformed image with the natural interpretation of vertical and horizontal edge bands. The second advantage of using the wavelet transform is the dramatically reduced computation that results from processing data at multiple scales [27]. The object search will later be formulated as an optimization problem in a high-dimensional space. The key to the efficient solution of this optimization will be a structured search that exploits the multiresolution structure of the wavelet transform.

Our wavelet decomposition uses the Haar basis functions illustrated in Fig. 5. Figure 6 shows an image resulting from this Haar wavelet decomposition. Notice that at each resolution, two of the bands are interpreted as the horizontal or vertical edge gradients. This structure will be used to make the image model sensitive to both region (average gray scale) and edge (gradient magnitude) information. Another advantage of the Haar basis functions is the computational simplicity resulting from coefficients of ±1.

We will generally assume that Y is the wavelet transformed image. The wavelet transform is an invertible, orthogonal transformation, so the transformed image contains all of the information in the original data. Furthermore, because the Jacobian

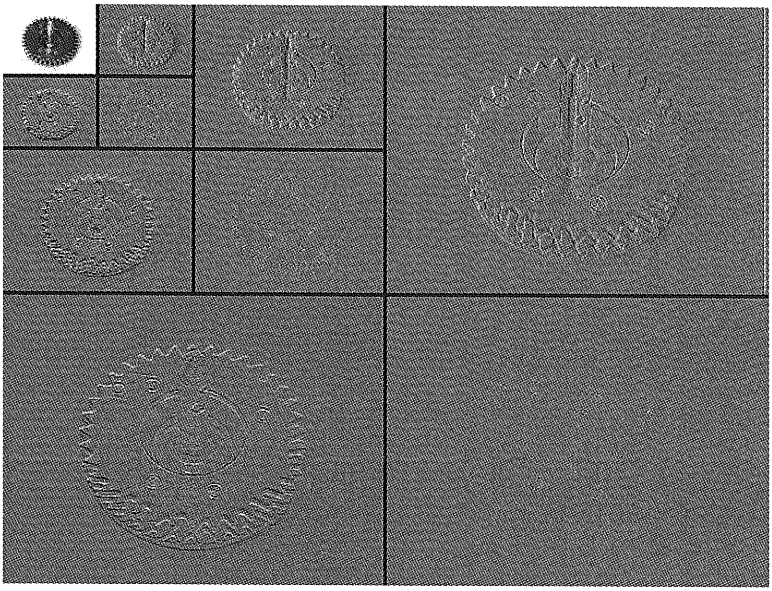


FIGURE 6 Wavelet decomposition using the Haar basis functions. The transformation generates separate vertical and horizontal bands at each resolution.

of the transformation is unity, the values of the density functions are equal for the original and transformed data.

For simplicity, our algorithm uses only the vertical and horizontal gradient information in the wavelet representation; we do not model the diagonal band information. This allows us to represent the data at each pixel location as a gradient vector. At each resolution l, define $Y_l = [Y_{lv}, Y_{lh}]$ where Y_{lv} and Y_{lh} are the vertical and horizontal bands of the wavelet transform. Generally, $0 \leq l \leq L-1$, where $l = 0$ is the finest resolution and $L-1$ is the coarsest. Each pixel in Y_l is denoted by $Y_l(i) = [Y_{lv}(i), Y_{lh}(i)]$, where $i = [i_1, i_2]^t$ is a vector index. Intuitively, this index corresponds to the physical position $[v, h] = [i_1 2^l + 2^{l-1}, i_2 2^l + 2^{l-1}]$.

The pixels $Y_l(i)$ are assumed to be conditionally independent, given the state $X^{(c)}$ and the data parameters θ. This is a reasonable assumption because the wavelet transform decorrelates the image data. Intuitively, the pixel value $y_l(i)$ represents the local gradient of the image at location i. Because image derivatives are known to be accurately modeled as Laplacian distributed [28], we choose a density function similar to the Laplacian density for our data distribution. In particular,

$$p\big(y_l(i) \mid X^{(c)} = x^{(c)}, \theta^{(c)}\big) = \frac{1}{2\pi(\lambda^{(c)}\tilde{\sigma}_l(i))^2} \exp\left\{-\frac{\|y_l(i) - \tilde{\mu}_l(i)\|}{\lambda^{(c)}\tilde{\sigma}_l(i)}\right\}, \tag{3}$$

where $\|\cdot\|$ is the Euclidean norm and $\tilde{\mu}_l(i)$ and $\tilde{\sigma}_l(i)$ are model parameters determined by $x^{(c)}$ and θ. The redundant parameter $\lambda^{(c)}$, which also depends on the state $x^{(c)}$ and the resolution l, has been added to explicitly account for local variation in image brightness. Note that this model differs slightly from the Laplacian density, which uses a 1-norm in place of the 2-norm.

The mean vector $\tilde{\mu}_l(i)$ of (3) is just the average local gradient at pixel location i. This characterizes gray-scale behavior, including edge polarity and sharpness. The variation parameters $\tilde{\sigma}_l(i)$ indicate the areas of greatest uncertainty in the template, which will generally occur near edges. Thus, the model is sensitive to both region-based and edge-based information, with the relative importance of each information type determined by the model parameters. Note that the variation parameter is common to both the vertical and horizontal wavelet bands. In this way a rotation of the subassembly can be modeled by simply rotating each mean vector $\tilde{\mu}_l(i)$.

To define the relationship between the parameters of (3) and $X^{(c)}$ and θ, we must first precisely define the components for θ. For node c of the object tree, the components for $\theta^{(c)}$ are $\theta_l^{(c)}(i) = [\mu_l(i), \sigma_l(i)]$, where $\mu_l(i) = [\mu_{lv}(i), \mu_{lh}(i)]$ is the average gradient at template location i, and i is a vector index that takes values in $W_l^{(c)}$. The set $W_l^{(c)}$ may be thought of as a window containing the subassembly in the normalized coordinate system. To eliminate spurious results due to insufficient data, we define $W_l^{(c)}$ to be empty for resolutions l at which this window contains fewer than 4×4 pixels. In Fig. 3, these windows correspond to the rectangular boxes.

The effect of the state $x^{(c)} = [(s^{(c)})^t, z^{(c)}, r^{(c)}]^t$ is to transform and distort the template of parameters $\theta^{(c)}$ and its associated window $W^{(c)}$. Therefore, to compute the parameters of a pixel we will determine the θ parameters that transform to the pixel location. Unfortunately, this coordinate transformation will generally yield noninteger positions in the coordinates of the template. We solve this problem by using bilinear interpolation to compute parameter values between grid points. The variation parameters form a scalar template that undergoes an affine transformation, and the mean vectors can be thought of as a local gradient field under the same transformation. The parameters of (3) are thus given by

$$\begin{aligned}\tilde{\mu}_l(i) &= \mu_l\left(\mathbf{T}^{(c)}\left(i-2^{-l}s^{(c)}\right)\right)\mathbf{T}^{(c)} \\ \tilde{\sigma}_l(i) &= \sigma_l\left(\mathbf{T}^{(c)}\left(i-2^{-l}s^{(c)}\right)\right),\end{aligned} \tag{4}$$

where the noninteger arguments of $\mu_l(\cdot)$ and $\sigma_l(\cdot)$ are interpreted as bilinear interpolation. Of course, (4) is only defined when i transforms to template locations contained in $W_l^{(c)}$. Therefore, this transformed window is defined to be

$$\widetilde{W}_l^{(c)} = \left\{i : \mathbf{T}^{(c)}\left(i-2^{-l}s^{(c)}\right) \in W_l^{(c)}\right\}.$$

Combining these ideas yields the complete data model at each resolution l,

$$p\left(y_l \mid X^{(c)} = x^{(c)}, \theta^{(c)}\right) = \prod_{i\in\widetilde{W}_l^{(c)}} \frac{1}{2\pi(\lambda^{(c)}\tilde{\sigma}_l(i))^2} \exp\left\{-\frac{\|y_l(i)-\tilde{\mu}_l(i)\|}{\lambda^{(c)}\tilde{\sigma}_l(i)}\right\}. \tag{5}$$

We should note that the model presented has a minor inconsistency. If the windows of the various subassemblies overlap, then there is more than one way in which the pixel parameters may be computed. Theoretically, this inconsistency could be eliminated by assigning a priority ordering to the nodes. For example, nodes closest to leaf nodes could occlude nodes higher in the tree. However, for computational simplicity we ignore this inconsistency and assume that the overlap of nodes in space and scale will not have a significant effect.

Also notice that pixels outside of the subassembly windows are not explicitly modeled. In practice, we will always compute ratios of density functions, so the contribution due to these unmodeled pixels will cancel out. Kopec and Chou use this same idea in their model for document images [15].

B. State Estimation

To compare a given image to our model, we must first locate each of the object subassemblies in the image. This is equivalent to estimating the four-dimensional state vector associated with each node of the object tree. We estimated the states with the use of the SMAP procedure of Bouman and Shapiro [10]. This technique simplifies the estimation problem by allowing the state of each node in the object tree to be estimated separately.

This section presents a multiscale technique for searching the state space for the most likely position and orientation of a subassembly. Because the search algorithm must be performed for every new image, it should be as efficient as possible. Computational efficiency is achieved by using the log likelihood at coarse resolutions to guide the search at finer resolutions.

The SMAP method starts at the object tree's root and progresses to its leaves. At each node of the tree, the maximum a posteriori (MAP) estimate of the state $X^{(c)}$ is computed, given the image data y and the estimated state at the parent node, $\hat{x}^{(p)}$. To simplify computation and avoid a recursive implementation, we modify the

SMAP algorithm by ignoring data terms from descendants of the node c. Under these assumptions, the SMAP state estimate for node c is given by

$$\hat{x}^{(c)} = \arg\max_{x^{(c)}} \left\{ \log p\left(y \mid X^{(c)} = x^{(c)}, \theta^{(c)}\right) + \log p\left(x^{(c)} \mid X^{(p)} = \hat{x}^{(p)}, \phi^{(c)}\right) \right\}.$$

To simplify computation, we use likelihood ratios to compute $\hat{x}^{(c)}$. Let $p_0(y)$ be some as yet undefined density function for the data when the subassembly c is not present. Note that because $p_0(y)$ does not depend on $X^{(c)}$,

$$\hat{x}^{(c)} = \arg\max_{x^{(c)}} \left\{ \log \frac{p\left(y \mid X^{(c)} = x^{(c)}, \theta^{(c)}\right)}{p_0(y)} + \log p\left(x^{(c)} \mid X^{(p)} = \hat{x}^{(p)}, \phi^{(c)}\right) \right\}. \quad (6)$$

A multiscale search procedure is used to perform the optimization in (6), so we need to define a multiresolution version of the expression in (6). With this in mind, the log likelihood ratio for resolutions coarser than l is defined to be

$$L(x^{(c)}, l) = \log \left(\prod_{m=l}^{L-1} \frac{p\left(y_m \mid X^{(c)} = x^{(c)}, \theta^{(c)}\right)}{p_0(y_m)} \right) + \log p\left(x^{(c)} \mid X^{(p)} = \hat{x}^{(p)}, \phi^{(c)}\right).$$

This expression, which we wish to maximize, is the sum of a data term and a prior term. The data term indicates how well the data at this state and resolution match the subassembly model. The prior term gives the prior likelihood of the subassembly appearing at this location and orientation.

The prior term of the log likelihood ratio is computed with the use of the prior state density function in (2), but the data term must still be precisely defined. For pixels $i \notin \widetilde{W}_l^{(c)}$, the presence or absence of the subassembly is irrelevant. Therefore, for all $i \notin \widetilde{W}_l^{(c)}$, $p_0(y_l(i)) = p(y_l(i) \mid X^{(c)} = x, \theta)$. If subassembly c is not present at state $x^{(c)}$, we have no a priori expectations for the pixel values in the window $\widetilde{W}_l^{(c)}$. We therefore assume that these pixels are independent and identically distributed. Because y_l is a bandpass signal with no DC component, we assume the values are zero mean with distribution

$$p_0(y_l) = \prod_{i \in \widetilde{W}_l^{(c)}} \frac{1}{2\pi\left(\lambda_0^{(c)}\right)^2} \exp\left\{ -\frac{\|y_l(i)\|}{\lambda_0^{(c)}} \right\}, \quad (7)$$

where $\lambda_0^{(c)}$ is the local average variation of the image data. Putting this model together with (5) yields the result

$$\log \left(\prod_{m=l}^{L-1} \frac{p\left(y_m \mid X^{(c)} = x^{(c)}, \theta^{(c)}\right)}{p_0(y_m)} \right)$$

$$= \sum_{m=l}^{L-1} \sum_{i \in \widetilde{W}_m^{(c)}} \left(2 \log \frac{\lambda_0^{(c)}}{\lambda^{(c)} \tilde{\sigma}_m(i)} - \frac{\|y_m(i) - \tilde{\mu}_m(i)\|}{\lambda^{(c)} \tilde{\sigma}_m(i)} + \frac{\|y_m(i)\|}{\lambda_0^{(c)}} \right). \quad (8)$$

We estimate the unknown parameter $\lambda_0^{(c)}$ by maximizing (7) with respect to this parameter, and the value of $\lambda^{(c)}$ is estimated by maximizing (5). This gives the final expression,

$$L(x^{(c)}, l) = 2\widehat{N}\log\frac{\hat{\lambda}_0^{(c)}}{\hat{\lambda}^{(c)}} - \sum_{m=l}^{L-1}\sum_{i\in\widetilde{W}_m^{(c)}} 2\log\tilde{\sigma}_m(i) + \log p\big(x^{(c)} \mid X^{(p)} = \hat{x}^{(p)}, \phi^{(c)}\big), \quad (9)$$

where

$$\hat{\lambda}_0^{(c)} = \frac{1}{2\widehat{N}}\sum_{m=l}^{L-1}\sum_{i\in\widetilde{W}_m^{(c)}} \|y_m(i)\| \hat{\lambda}^{(c)} = \frac{1}{2\widehat{N}}\sum_{m=l}^{L-1}\sum_{i\in\widetilde{W}_m^{(c)}} \frac{\|y_m(i) - \tilde{\mu}_m(i)\|}{\tilde{\sigma}_m(i)}$$

$$\widehat{N} = \sum_{m=l}^{L-1}\sum_{i\in\widetilde{W}_m^{(c)}} 1.$$

Note that the estimates $\hat{\lambda}_0^{(c)}$ and $\hat{\lambda}^{(c)}$ depend on the resolution l and the subassembly state $x^{(c)}$, which determines the windows $\widetilde{W}_m^{(c)}$. The log likelihood ratio in (9) can now be computed at any candidate state $X^{(c)} = x^{(c)}$ and resolution l.

We next devise a procedure for searching the states, $x^{(c)}$, and resolutions, l, in an efficient manner. The possible subassembly positions, $x^{(c)}$, must be sampled at discrete points, and computation is saved by sampling $x^{(c)}$ more coarsely for large values of l corresponding to coarse resolution. Rotation and scale changes should also be sampled more finely for large templates. To do this, define the constant $d^{(c)}$ to be the diameter of the smallest circle containing the template at scale factor $z^{(c)} = 1$ and resolution $l = 0$. Then the sampling period of $z^{(c)}$ and $r^{(c)}$ should be inversely proportional to $d^{(c)}$. Using this approach, define $k = [k_1, k_2, k_3, k_4]^t$ to be a vector of integer indices, and let $x(k, l)$ be the vector function

$$x(k, l) = \left[k_1 2^{l-1} + 2^{l-2},\ k_2 2^{l-1} + 2^{l-2},\ \frac{k_3 2^l + 2^{l-1}}{d^{(c)}},\ \frac{k_4 2^l + 2^{l-1}}{d^{(c)}}\right]^t.$$

The function $x(k, l)$ gives the candidate states at each resolution l, which we link to those at the next finer resolution by defining the neighbors of (k, l) to be

$$\text{next}(k, l) = \big\{(n, l-1) \mid n_i = 2k_i \text{ or } n_i = 2k_i + 1\big\}.$$

The state indices k_1, k_2, k_3, and k_4 correspond to vertical and horizontal position, scale factor, and rotation angle, respectively. The index k_3 must therefore be nonnegative because only positive scale factors are possible, and the rotation angle must be between $-\pi$ and π, setting limits on the possible values of k_4 at each resolution l. The vertical and horizontal positions are nominally unconstrained, although in practice indices k_1 and k_2 are limited such that the position falls within the image boundaries. Figure 7 illustrates this sampling scheme for a single state component. Note that the candidate states form a binary tree that densely samples the space of possible states.

The multiscale search procedure is defined on this tree structure, and it proceeds based on the log likelihood ratio $L_d(k, l) \equiv L(x(k, l), l)$ associated with each sampling index k and resolution l. We initialize the search for a subassembly c by computing the log likelihood ratios over all vector indices $k \in \mathscr{X}^{(c)}(\alpha, l)$, where

$$\mathscr{X}^{(c)}(\alpha, l) = \Big\{k : \log p\big(x(k, l) \mid X^{(p)} = \hat{x}^{(p)}, \phi^{(c)}\big) > \alpha\Big\},$$

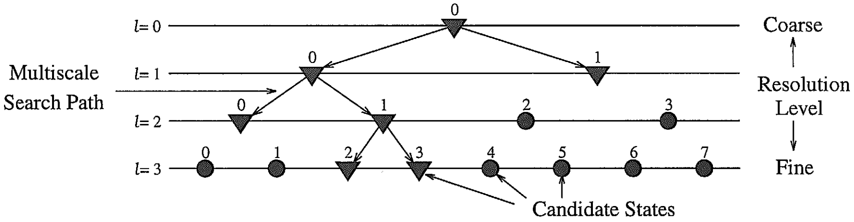


FIGURE 7 Multiscale sampling for a one-dimensional state space. The index k associated with each sample is as labeled. A multiscale search procedure is carried out on these samples to compute the state estimate.

and α is a user-defined rejection threshold. The initialization takes place at resolution $l = \max(l_{M_0}, l_0^{(c)})$, where l_{M_0} is equal to the coarsest resolution at which $\mathcal{X}^{(c)}(\alpha, \cdot)$ contains at least M_0 elements, and $l_0^{(c)}$ is the finest resolution at which the search is permitted to proceed. The constant M_0 is used to make sure the search is initialized with a reasonable number of points, and the finest resolution $l_0^{(c)}$ is set during training with the use of a heuristic procedure described in [22].

The initial candidate states and their associated log likelihood ratios are stored in a data structure known as a heap. This structure allows efficient insertion of new values and extraction of the pairs (k, l) with the largest log likelihood ratios.

After initialization, the search locates the M most promising search paths and expands them to the next finer resolution by computing the log likelihood ratios $L_d(\cdot)$ associated with their neighbors. If any of these log likelihood ratios fall below a rejection threshold α, the algorithm discards the corresponding state, thereby pruning the search space. If any of the log likelihood ratios exceed an acceptance threshold β, the corresponding state is returned as the state estimate $\hat{x}^{(c)}$. Candidate states with $L_d(\cdot)$ between α and β are stored on the heap. The algorithm then extracts the M best states from the updated heap and the process repeats. Because the best candidate states can occur at any resolution, the multiscale search can backtrack to coarser resolutions if necessary to investigate additional search paths. We improve robustness by choosing $M > 1$ and investigating multiple search paths simultaneously.

As illustrated in Fig. 8, the search takes the form of a sequential likelihood ratio test in which β and α represent acceptance and rejection thresholds. If these

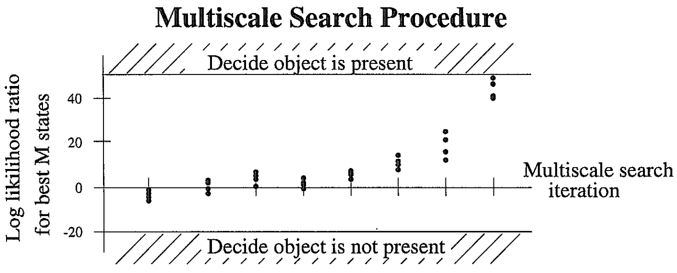


FIGURE 8 An example search procedure for $M = 4$. The search terminates when it encounters a candidate state whose log likelihood ratio exceeds β or when the heap has been exhausted (all remaining candidate states have log likelihood ratios less than α).

```
1. set $l = \max\left(l_{M_0}, l_0^{(c)}\right)$
2. for all $k \in \chi^{(c)}(\alpha, l)$
3.     compute $L(k, l)$ and store $(k, l)$ on the heap
4. while the heap is nonempty
5.     extract the $M$ largest likelihood ratios $L\left(k_{(1)}, l_{(1)}\right) \ldots L\left(k_{(M)}, l_{(M)}\right)$ from
       the heap
6.     if for all $i, l_{(i)} == l_0^{(c)}$
7.         if $L\left(k_{(1)}, l_{(1)}\right) > \beta_0$ stop with match $\left(k_{(1)}, l_{(1)}\right)$
8.         else stop with no match
9.     for $i = 1$ to $M$
10.        if $l_{(i)} > l_0^{(c)}$
11.            for all $(k, l) \in \text{next}\left(k_{(i)}, l_{(i)}\right)$
12.                compute $L(k, l)$
13.                if $L(k, l) > \beta$, stop with match $(k, l)$
14.                if $L(k, l) > \alpha$, store $(k, l)$ on the heap
15.        else
16.            store $\left(k_{(i)}, l_{(i)}\right)$ on the heap
17. stop with no match
```

FIGURE 9 Multiscale search algorithm for inspection. Lines 1–3 initialize the heap data structure. Lines 6–8 check to see if all candidate nodes are at the finest resolution and, if they are, compare the maximum ratio to β_0. Lines 10–16 search children of candidate nodes.

thresholds are not exceeded, the search process continues to finer resolutions, where more data are obtained. If the search reaches a point at which all M candidate states are at the finest resolution, then a decision is made by comparing the log likelihood to a third threshold, β_0.

The search is implemented as described in Fig. 9. For our experiments we use the values $\alpha = -15$, $\beta = 100$, $\beta_0 = 20$, $M = 16$, and $M_0 = 100$. If the search for a particular subassembly terminates in a rejection (no match), that subassembly is declared missing, and the SMAP procedure is terminated for descendents of that node.

In some cases this search procedure will terminate with a match at a resolution $l^{(p)} > l_0^{(p)}$ for node p. The resulting coarse state estimate $\widehat{X}^{(p)}$ can be viewed as a quantized version of the actual state, which we take to be at resolution $l_0^{(p)}$, so

$$\widehat{X}^{(p)} = X^{(p)} + Q.$$

This quantization error will increase the uncertainty in the location of subassembly c, a child node of p. This increased uncertainty is accounted for by changing the covariance matrix of (2) to

$$\widetilde{\mathbf{B}} = \mathbf{B} + \mathbf{B}_Q(\widehat{X}^{(p)}, \phi^{(c)}, l^{(p)}, l_0^{(p)}),$$

where $\mathbf{B}_Q(\cdot)$ is a diagonal matrix computed in [22].

C. Training Algorithm (Parameter Estimation)

An iterative procedure based on the EM algorithm is used to estimate the model parameters θ and ϕ from a set of training images. The first training image $Y(\cdot, 0)$ is distinct from the rest because the states, X, are assumed to be known. This image, which we will refer to as the initialization image, defines the regions associated with each subassembly and will also be used to initialize the model parameters. A nominal image of this sort can easily be generated from the CAD model of the assembly.

Ideally, we would like to compute the maximum likelihood estimates of θ and ϕ, given the N training images $Y(\cdot, 0) \cdots Y(\cdot, N-1)$. However, this would require a joint optimization over the entire object tree, which is too computationally complex. Instead, the estimates of $\theta^{(c)}$ and $\phi^{(c)}$ are computed at each node c with the use of the N images and $\hat{x}_n^{(p)}$, the estimated parent state for image n:

$$(\hat{\theta}^{(c)}, \hat{\phi}^{(c)}) = \arg \max_{(\theta^{(c)}, \phi^{(c)})} \prod_{n=0}^{N-1} p(y(\cdot, n) \mid X_n^{(p)} = \hat{x}_n^{(p)}, \theta^{(c)}, \phi^{(c)}). \tag{10}$$

As with the SMAP state estimation of the previous subsection, data from descendants of node c are ignored.

Notice that (10) may be implemented as a sequence of optimizations at individual nodes. Because each optimization depends on the estimated parent states $\hat{x}_n^{(p)}$, this sequence must proceed in order from root to leaves.

The difficulty in computing (10) is the missing state information $X_n^{(c)}$. Without this state at each image, we cannot determine the best state parameters $\phi^{(c)}$ or the template parameters $\theta^{(c)}$. The EM algorithm is specifically formulated to solve such "missing data" problems.

The EM algorithm works by computing a sequence of parameter estimates that converge to a local maximum of (10). The EM update equation is given by

$$(\hat{\theta}_{\text{new}}^{(c)}, \hat{\phi}_{\text{new}}^{(c)}) = \arg \max_{(\theta^{(c)}, \phi^{(c)})} \sum_{n=0}^{N-1} E\big[\log p\big(y(\cdot, n), X_n^{(c)} \mid X_n^{(p)} = \hat{x}_n^{(p)}, \theta^{(c)}, \phi^{(c)}\big) \,\big|\, \Gamma_n\big],$$

where

$$\Gamma_n = \big\{Y(\cdot, n) = y(\cdot, n), X_n^{(p)} = \hat{x}_n^{(p)}, \hat{\theta}_{\text{old}}^{(c)}, \hat{\phi}_{\text{old}}^{(c)}\big\},$$

and $\hat{\theta}_{\text{old}}^{(c)}$ and $\hat{\phi}_{\text{old}}^{(c)}$ are the parameters from the previous iteration. Using Bayes's rule and noting that data parameters must be estimated for all subassembly resolutions $\leq l_0^{(c)}$, we get two separate update equations,

$$\hat{\theta}_{\text{new}}^{(c)} = \arg\max_{\theta^{(c)}} \sum_{n=0}^{N-1} \sum_{l=l_0^{(c)}}^{L-1} E\big[\log p(y_l(\cdot, n) \mid X_n^{(c)}, \theta^{(c)}) \mid \Gamma_n\big] \tag{11}$$

$$\hat{\phi}_{\text{new}}^{(c)} = \arg\max_{\phi^{(c)}} \sum_{n=0}^{N-1} E\big[\log\big(X_n^{(c)} \mid X_n^{(p)} = \hat{x}_n^{(p)}, \phi^{(c)}\big) \,\big|\, \Gamma_n\big]. \tag{12}$$

Consider the state parameter update of (12). The update equations for the components of $\phi^{(c)} = [(m^{(c)})^t, (\gamma^{(c)})^t]^t$ can be computed by using the prior state density in (2) and then setting the derivative with respect to $\phi^{(c)}$ to zero. The update for the state means is given by

$$\hat{m}^{(c)}_{\text{new}} = \mathbf{A}^{-1}\frac{1}{N}\sum_{n=0}^{N-1} E\big[\big(X^{(c)}_n - \hat{x}^{(p)}_n\big)\,|\,\Gamma_n\big].$$

The EM update equations will all contain expected values over the posterior state density for node c in each training image. Each of these expectations can be approximated as a weighted sum over the sampled states at resolution $l^{(c)}_0$, but the values associated with the most likely state typically dominate this sum by orders of magnitude. We therefore approximate the expected values by the values corresponding to the most likely state, which is the state found by the multiscale search procedure. This same approach is often taken when analogous expressions in speech and text recognition are being solved [15]. The update equation for the state means is then given by

$$\hat{m}^{(c)}_{\text{new}} = \mathbf{A}^{-1}\frac{1}{N}\sum_{n=0}^{N-1}\big(\hat{x}^{(c)}_n - \hat{x}^{(p)}_n\big). \tag{13}$$

A similar method is used to compute the updates for the variance parameters $\gamma^{(c)}$. These updates are given by

$$\left[\hat{\gamma}^{(c)}_{s,\text{new}}, \hat{\gamma}^{(c)}_{z,\text{new}}, \hat{\gamma}^{(c)}_{r,\text{new}}\right]^t = \left[\frac{1}{2N}\sum_{n=0}^{N-1}\frac{\|\delta_s(n)\|^2}{\big(\hat{z}^{(p)}_n\big)^2}, \frac{1}{N}\sum_{n=0}^{N-1}\delta^2_z(n), \frac{1}{N}\sum_{n=0}^{N-1}\delta^2_r(n)\right]^t,$$

where

$$\left[\delta_s(n), \delta_z(n), \delta_r(n)\right]^t = \hat{x}^{(c)}_n - \hat{x}^{(p)}_n - \mathbf{A}\hat{m}^{(c)}_{\text{new}}.$$

Now we need to compute the update equations for the data parameters from (11). Recall that a parameter $\lambda^{(c)}$ is used in the data model to account for intensity scaling of image regions, which is necessary for the log likelihood ratio computations. During training, however, all data variability among the training images is incorporated into the variability parameter estimates, $\tilde{\sigma}_l(\cdot)$, so $\lambda^{(c)}$ becomes an arbitrary constant, which we set to one.

The template components $\mu_l(\cdot)$ and $\sigma_l(\cdot)$ can be expressed in terms of the parameters $\tilde{\mu}_l(\cdot)$ and $\tilde{\sigma}_l(\cdot)$ of Eq. (5) by performing the inverse of the transformations in (4). However, the transformations of (4) may not be strictly invertible, because the size of the transformed window $\widetilde{W}^{(c)}_l$ may not be the same as the size of the untransformed window $W^{(c)}_l$. We avoid this problem by using bilinear interpolation on the data values to approximate the inverse of the bilinear interpolation in (4). Because each expectation in (11) is approximated by the value at the most likely state $\hat{x}^{(c)}_n = [(\hat{s}^{(c)}_n)^t, \hat{z}^{(c)}_n, \hat{r}^{(c)}_n]^t$ for each training image n, the template components are computed as

$$\mu_l(i) \approx \tilde{\mu}_l\big((\mathbf{T}^{(c)}_n)^{-1}i + 2^{-l}\hat{s}^{(c)}_n, n\big)(\mathbf{T}^{(c)}_n)^{-1}$$
$$\sigma_l(i) \approx \tilde{\sigma}_l\big((\mathbf{T}^{(c)}_n)^{-1}i + 2^{-l}\hat{s}^{(c)}_n, n\big),$$

where $[\tilde{\mu}_l(i,n), \tilde{\sigma}_l(i,n)]^t$ are the parameters corresponding to pixel $y_l(i,n)$ of training image n, and $\mathbf{T}_n^{(c)}$ is the transformation matrix evaluated at $\hat{x}_n^{(c)}$.

The image pixel values at the template component locations can be approximated via the same transformation. Let

$$\tilde{y}_l(i,n) = y_l\big((\mathbf{T}_n^{(c)})^{-1} i + 2^{-l}\hat{s}_n^{(c)}, n\big)(\mathbf{T}_n^{(c)})^{-1}.$$

Each expectation in (11) can now be approximated by the value at the most likely state $\hat{x}_n^{(c)}$, yielding a sum over the pixels in the window $\widetilde{W}_l^{(c)}$. If this sum is thought of as an approximation to an integral over the window, a simple change of variables leads to a second approximation as a sum over the untransformed window $W_l^{(c)}$. Ignoring terms do not depend on $\theta^{(c)}$, this gives

$$\begin{aligned} &E\big[\log p\big(y_l(\cdot,n) \mid X_n^{(c)}, \hat{\theta}^{(c)}\big) \mid \Gamma_n\big] \\ &\approx \sum_{i \in \widetilde{W}_l^{(c)}} \left\{ -2\log(\lambda^{(c)}\tilde{\sigma}_l(i,n)) - \frac{\|y_l(i,n) - \tilde{\mu}_l(i,n)\|}{\lambda^{(c)}\tilde{\sigma}_l(i,n)} \right\} \\ &\approx |(\mathbf{T}_n^{(c)})^{-1}| \sum_{i \in W_l^{(c)}} \left\{ -2\log\sigma_l(i) - \frac{\|\tilde{y}_l(i,n)(\mathbf{T}_n^{(c)})^{-1} - \mu_l(i)(\mathbf{T}_n^{(c)})^{-1}\|}{\sigma_l(i)} \right\} \\ &= (\hat{z}_n^{(c)})^2 \sum_{i \in W_l^{(c)}} \left\{ -2\log\sigma_l(i) - \frac{\hat{z}_n^{(c)}\,\|\tilde{y}_l(i,n) - \mu_l(i)\|}{\sigma_l(i)} \right\}, \end{aligned} \tag{14}$$

where the invariance of the 2-norm under rotation is used to obtain the final expression.

Substituting (14) into (11), the EM updates for the template parameters are given by

$$\hat{\mu}_{l,\text{new}}(i) = \arg\min_{\mu_l(i)} \sum_{n=0}^{N-1} (\hat{z}_n^{(c)})^3 \|\tilde{y}_l(i,n) - \mu_l(i)\| \tag{15}$$

$$\hat{\sigma}_{l,\text{new}}(i) = \frac{\sum_{n=0}^{N-1} (\hat{z}_n^{(c)})^3 \|\tilde{y}_l(i,n) - \mu_{l,\text{new}}(i)\|}{2\sum_{n=0}^{N-1} (\hat{z}_n^{(c)})^2}. \tag{16}$$

The computation of (15) would require a recursive implementation, so we approximate the update by assuming that the scale factors are all near unity and replacing the 2-norm with a 1-norm. This gives the update

$$\hat{\mu}_{l,\text{new}}(i) = \text{Median } \{\tilde{y}_l(i,0), \ldots, \tilde{y}_l(i,N-1)\}.$$

Because the EM algorithm is only guaranteed to converge to a local maximum of the likelihood equation, the final estimates can vary considerably, depending on the initial starting point. We have devised a heuristic technique to compute initial parameter estimates [22].

The EM update scheme for a subassembly proceeds as shown in Fig. 10. We set $N_{\text{EM}} = 4$. The algorithm tends to converge to a fairly stable set of parameters by this point. Note that the finest model resolution $l_0^{(c)}$ is set to 0 for leaf nodes.

```
initialize parameter estimates $\hat{\theta}^{(c)}$ and $\hat{\phi}^{(c)}$
initialize $l_0^{(c)} = L - 1$
set $EM\text{-}iteration = 0$
while $EM\text{-}iteration < N_{EM}$ and $\left(\hat{\theta}_{new}^{(c)}, \hat{\phi}_{new}^{(c)}\right) \neq \left(\hat{\theta}_{old}^{(c)}, \hat{\phi}_{old}^{(c)}\right)$
    for $n = 1$ to $N - 1$
        use multiscale search to compute state estimates $\hat{x}_n^{(c)}$ at resolution
        $l_n \leq l_0^{(c)}$
    if $EM\text{-}iteration == 0$ and $c$ is a leaf node, set $l_0^{(c)} = l = 0$
    else set $l = \max_{n>0} l_n$
    if $l > 0$, update $\hat{\theta}^{(c)}$ to resolution $l - 1$ using EM update equations
    else update $\hat{\theta}^{(c)}$ to resolution $l$ using EM update equations
    if $EM\text{-}iteration > 0$
        set $l_0^{(c)} = l$
        update $\hat{\phi}^{(c)}$ using EM update equations
multiply $\hat{\gamma}_s^{(c)}$, $\hat{\gamma}_z^{(c)}$, and $\hat{\gamma}_r^{(c)}$ by $\left(\frac{2N}{2N-1}\right)$, $\left(\frac{N}{N-1}\right)$ and $\left(\frac{N}{N-1}\right)$, respectively,
to remove bias
store $\hat{\theta}^{(c)}$ and $\hat{\phi}^{(c)}$ as model parameters for this subassembly, with $l_0^{(c)}$ as the
finest model resolution
```

FIGURE 10 EM algorithm for training. This procedure adapts the model to the variations seen in the training set.

These nodes are normally associated with subassemblies that are important for proper detection, so we force the algorithm to model these subassemblies at the finest resolution. The finest resolution for other nodes is initialized to $L - 1$ and is set in a monotonically nonincreasing fashion during the training procedure.

The multiscale search procedure used during the training phase is similar to the procedure used when an image is tested, but several changes had to be made to ensure that the search will terminate in a match, because all training images contain properly assembled objects by definition. The differences are detailed in [22].

In general, we have found this algorithm to be quite robust, although it can encounter difficulty in trying to detect small features that have no sharp edges, particularly if they lie in areas of high activity in the image. This tendency can be reduced to some extent by the use of feature-shaped subassembly windows, but for small features this will reduce the number of pixels in the window even more, and any occlusion problems will remain. The algorithm also tends to overestimate the scale factor for small features.

Our algorithm implicitly assumes that the large majority of test images will contain correctly assembled objects. For a misassembled object, the search must discard all candidate search paths before it can terminate with no match. Thus, the amount of computation required for an object with assembly errors is typically much greater than that required for a correctly assembled object. However, we do not consider this to be a problem because for our application, most of the inspected objects should be correctly assembled.

III. AUTOMATED CAMERA AND LIGHT PLACEMENT

The previous section illustrated how our multiscale detection algorithm is based on a stochastic object model, which is tailored to a specific assembly by adjustment of the model structure and changes in model parameters. The model generation and parameter estimation are driven by a CAD model of the assembly. The CAD information of the assembly, especially that of components of interest where errors are expected to occur, was used to identify the object tree and set the model parameters of the inspection algorithm [29]. Because it is assumed that all components are assembled only by vertical insertion, the components of interest are usually the ones being inserted. For example, Fig. 11 shows an exploded view of a typical mechanical assembly, and Fig. 12 shows an object tree that was calculated for that assembly. In this section we discuss how CAD information about the components of interest in the object tree is used to automatically set the camera and light source parameters to optimize the performance of the inspection algorithm.

This section is organized as follows. The issues related to the rendering techniques that we used are addressed in Section III.A. The camera placement algorithm is then described in Section III.B, followed by a description of the light placement algorithm in Section III.C. Finally, the generate-and-test approach is outlined in Section III.D.

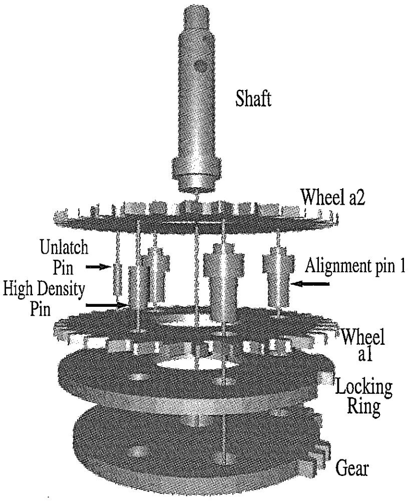


FIGURE 11 An exploded view of a typical mechanical assembly generated from the information in the CAD model. This view illustrates the order of assembly as well as the single common axis of insertion for all of the pins.

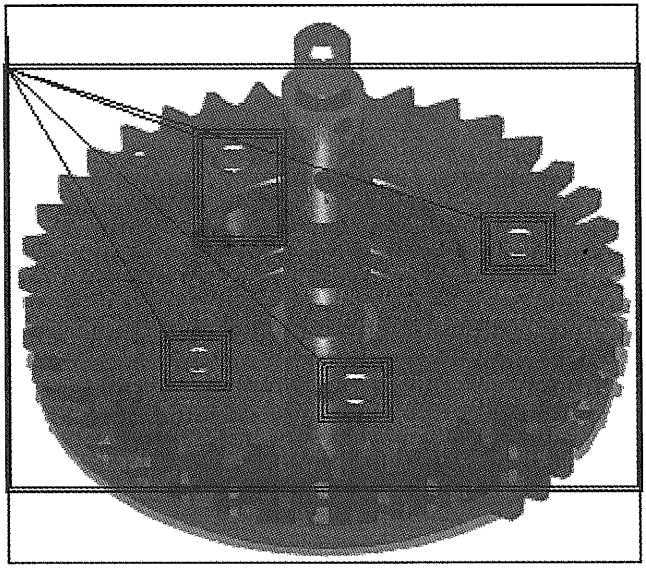


FIGURE 12 A synthetic image of a pattern wheel assembly with an object tree (denoted by the connected boxes) calculated using the CAD information of the inserted pins. This tree is required by the inspection algorithm to guide its analysis of the image. The number of boxes around each object represents the object's level in the tree. The boxes are automatically generated by calculating the visible portions of the components in the tree, with the first level box including the entire assembly.

A. Synthetic Image Generation

There are two image generation algorithms used to create synthetic images from the CAD model of the assembly. The first, addressed in Section III.A.1, is a fast rendering technique that uses only a simple local illumination model and takes advantage of special-purpose Very Large Scale Integrated (VLSI) hardware for performing geometrical calculations. This rendering process is important for the automated camera and light source placement process. The second rendering technique, addressed in Section III.A.2, is used to create more physically realistic synthetic images that are required to build the statistical model of what a correctly assembled product should look like. It is also used by the algorithm that determines an optimal light source location. The image rendering process used for these last two applications must simulate reality as closely as possible.

1. Fast Rendering Algorithm

Fast rendering algorithms running on special-purpose graphics workstations are used to create draft images of the assembly. These draft images are used to accomplish two main tasks. The first is to further refine the object trees used by the inspection algorithm. This is done by creating a mask for each of the rectangular object nodes. This mask is used to identify the regions within the node that correspond to related component surfaces, with only this region being used for building the statistical model of the node. This prevents irrelevant background information from affecting the sensitivity of the inspection process. The second purpose of these draft images is to identify the visible faces on the different assembly components for use by the algorithms that identify the optimal camera and light source location (described in Sections III.B and III.C). To do this, each of the surfaces of a main component of interest in the object trees is tagged with a unique ambient color, with all other surfaces of the other components set to black. The assembly is then rendered on a graphics workstation equipped with a Z-buffer, using only the ambient intensity of the polygons. The resulting image contains the number of visible pixels for each surface of interest, as well as providing information for the object node mask. Figure 13 illustrates this procedure.

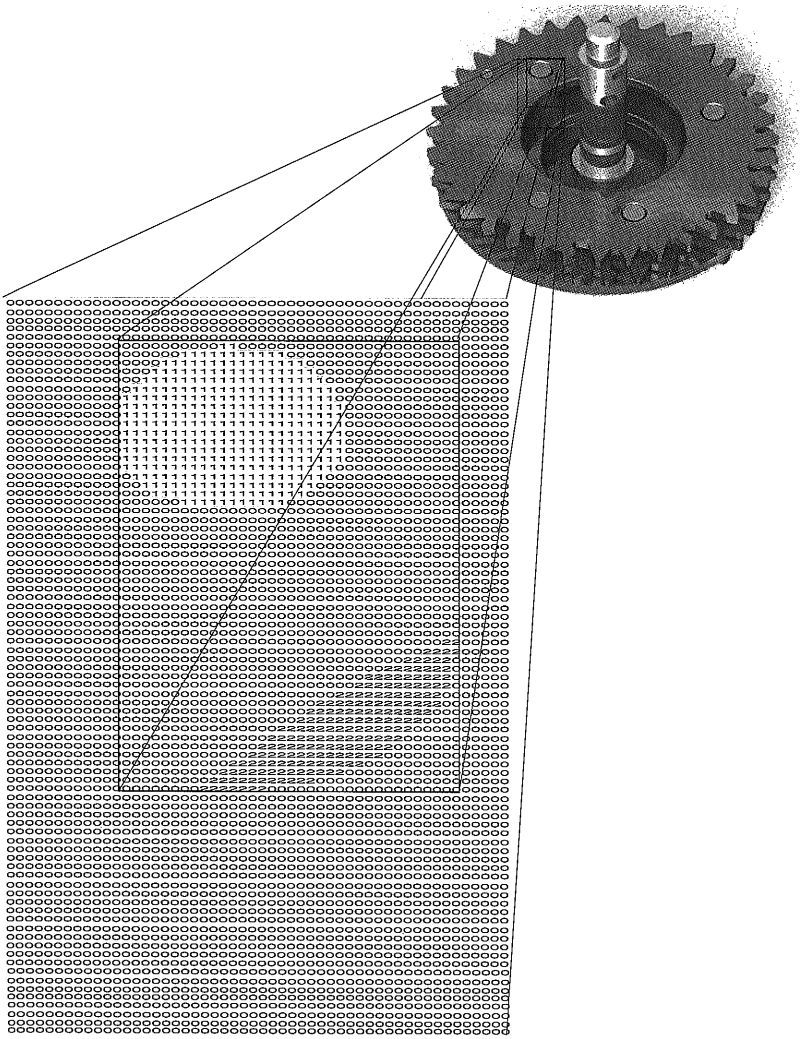


FIGURE 13 The outer rectangle represents the bounding box of the projection of an alignment pin in the assembly onto the image plane. The inner rectangle is the bounding box of the visible portion of this alignment pin. This bounding box is passed to the inspection algorithm as an object node along with the mask that identifies the region that corresponds to the alignment pin. Also, visible faces of the component are identified along with the amount visible. This information is obtained using Z-buffer hardware.

2. Accurate Rendering Algorithm

More accurate graphic rendering techniques are required to generate realistic synthetic training images of an assembly for building the statistical model of correctly assembled components. To obtain sufficiently realistic images, light–object interaction must be modeled. Although graphics workstations that are available today can generate shaded images at video rates, the illumination models used to generate these images typically only deal with the very first reflection from an object's surface. These so-called first-order or local models do not include light effects caused by light reflecting from several objects or being transmitted through objects. These global models need to be considered to obtain more realistic images that can model different types of materials, particularly those that are highly reflective, such as polished metals. The only established rendering techniques that attempt to model global lighting effects are ray tracing and radiosity. Because specular effects are very hard to model with radiosity, metallic parts are hard to simulate in images that are rendered with radiosity techniques [30]. As a result, ray tracing is selected as the rendering technique for this application.

Ray tracing as a comprehensive rendering technique was presented in [31]. The algorithm calculates the light reaching the eye from the scene by firing rays from the eye through each pixel in the image plane and tracing them through the scene (see Fig. 14). Each fired ray is checked for an intersection with the objects in the scene, and the intersection point closest to the eye is identified for each ray. A linear combination of three terms is used to calculate the intensity of the light reflected from these intersection points;

$$I = I_{\mathrm{local}} + k_{\mathrm{rg}} I_{\mathrm{r}} + k_{\mathrm{tg}} I_{\mathrm{t}}, \tag{17}$$

where I_{local} is the intensity of light energy reflecting at the intersection point directly from the light sources; I_{r} is the intensity of light energy arriving along the perfect specular direction ($\hat{R}$) from other surfaces in the environment; I_{t} is the intensity of light energy transmitted through the intersection point in a direction (obeying Snell's law) from other surfaces in the environment; k_{rg} is the global specular bidirectional reflectance coefficient; and k_{tg} is the global bidirectional transmission coefficient.

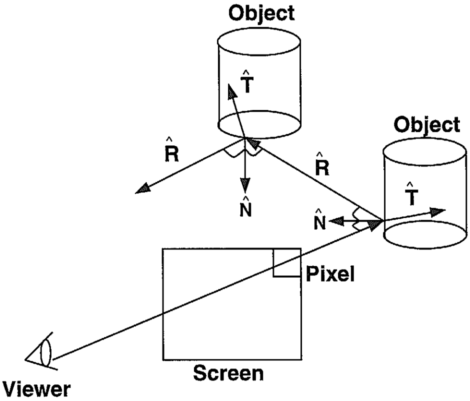


FIGURE 14 The raytracing algorithm.

The intensities I_r and I_t are calculated recursively by firing rays in the reflection and refraction directions from every intersection point. Intersections are again calculated for these rays, and Eq. (17) is reapplied.

I_{local} involves modeling the reflection of light energy from visible surface points. In his original paper on raytracing, Whitted used a Lambertian model to calculate I_{local}. In this work, the Cook–Torrance lighting model [32], a more accurate lighting model based on geometrical optics is used to calculate I_{local} in creating the desired realistic images. This led to a better match between real and synthetic images, which improved the training process. In addition, experiments comparing the inspection algorithm training and testing on real images of an assembly versus only synthetic images showed a high degree of correspondence.

B. Camera Placement

The CAD information guides the inspection algorithm in the training process by generating synthetic images that address different possible variations. Many factors affect the performance of an inspection algorithm. The inspection algorithm used in this work relies on gray-scale images. As a result, it relies heavily on the visible surfaces in the areas of interest and the gray-scale intensities associated with these areas. Therefore, it is essential to consider the effect of the viewing and light source parameters on the performance of the inspection algorithm. The viewing parameters considered are the camera's location E (the eye point); the center of interest C, which is a point along the camera's view direction; and the camera's field of view α, which is an angle that identifies the region in front of the camera that will be projected on the image plane. This section will address the issue of utilizing the CAD model to optimize these parameters.

If the entire assembly is to be tested for errors, then it is essential to have the entire assembly within the field of view. As a result, an approach similar to the one taken in [4, 5] is adopted. The assembly is assumed to lie on a planar surface. The camera and light source locations are restricted to lie within the surface of a hemisphere that surrounds the assembly, with the circular base of the hemisphere lying on the planar surface. The center of interest is constrained to the center of the circular base of the hemisphere. If a unit vector $\hat{u}$ that specifies a viewing direction from the eye point to the center of interest is found, then the eye point is placed such that the entire assembly is guaranteed to be in view. If the radius of the bounding hemisphere is r and the center of interest is the vector C, then the eye point location E is chosen to be

$$E = C - \frac{2r}{\alpha}\hat{u}. \tag{18}$$

Next, the viewing direction $\hat{u}$ is analytically limited to a region satisfying a separation requirement among the different assembly components of interest. Then, a function that evaluates the quality of a camera location is created. This function is used during the generate-and-test phase of the automation process that is described in Section III.D.

I. The Analytical Constraint

The contact information among the different components of the assembly is used to determine areas of interest within the image [29]. This contact information is collected for a specified group of components of interest. Maintaining a spatial separation between these components within the image improves the performance of the algorithm. Thus, one would like the distances between these components in the image to be as close as possible to their true lengths. The distances between the different components is calculated to form a weighted full graph in which each vertex is represented by a component. Each edge of the full graph is represented by the vector of minimum magnitude that separates the two components that are associated with the vertices of that edge. The magnitude of this vector is considered as the edge weight in the weighted full graph. So, the task becomes to view the edges of the full graph as close as possible to their true lengths.

To view the edges of the full graph as close as possible to their true lengths, singular-value decomposition (SVD) is used. It finds the dominant directions among all of the different vectors. Emphasis can be placed on smaller distances by modifying the magnitude of each vector to be the inverse of its original magnitude. So, if the full graph has n vertices, then $e = (n^2 - n)/2$ vectors are created. A real matrix $\mathbf{A}$ is formulated as follows:

$$\mathbf{A} = \begin{bmatrix} \frac{x_1}{m_1^2} & \frac{y_1}{m_1^2} & \frac{z_1}{m_1^2} \\ \frac{x_2}{m_2^2} & \frac{y_2}{m_2^2} & \frac{z_2}{m_2^2} \\ \vdots & \vdots & \vdots \\ \frac{x_e}{m_e^2} & \frac{y_e}{m_e^2} & \frac{z_e}{m_e^2} \end{bmatrix} \in R^{e\times 3}, \tag{19}$$

where m_i is the magnitude of vector i. The SVD of A,

$$A = U\Sigma V^T, \tag{20}$$

is then calculated, where U and V are orthogonal and

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix} \in R^{e\times 3}, \tag{21}$$

with $\sigma_1 > \sigma_2 > \sigma_3$. In addition,

$$V = \begin{bmatrix} \hat{v}_1 & \hat{v}_2 & \hat{v}_3 \end{bmatrix} \in R^{3\times 3}. \tag{22}$$

The resulting $\hat{v}_3$ forms the view direction from which the full graph edges will be seen as close as possible to their true lengths. However, this view direction does not address the occlusion problem. Therefore, it is essential to search off the $\hat{v}_3$ direction, yet keeping as much of the gained spread view as possible. This

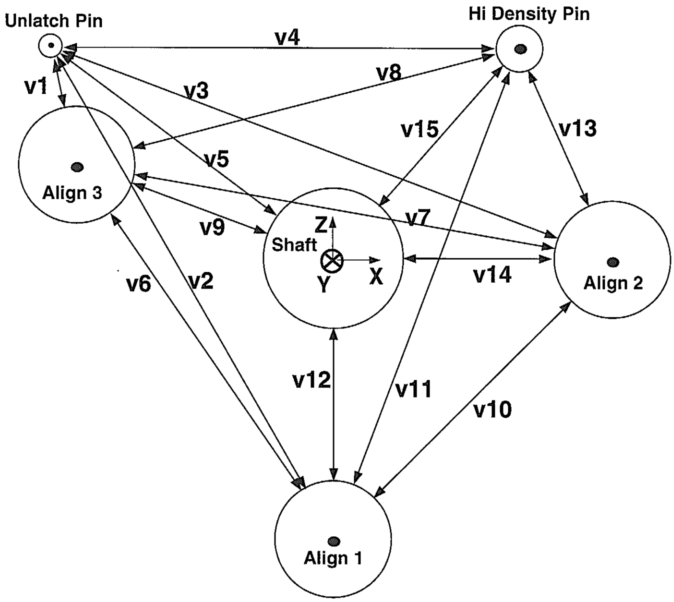


FIGURE 15 The full graph separating the shaft and the pins of the pattern wheel assembly, shown to scale.

is accomplished by limiting the camera to lie in the $\hat{v}_2$ $\hat{v}_3$ plane. This would constrain the camera to lie within a semicircle on the hemisphere. A generate-and-test approach is then used to maximize a function on the semicircle.

As an example, consider the pattern wheel assembly shown in Fig. 11. One can identify the shaft and the pins as components that are to be inserted. Figure 15 shows the full graph of the shaft and the pins. After the SVD algorithm described above was run, $\hat{v}_2$ was used to look at the shaft and the pins producing Fig. 16a. It is interesting to note how close the SVD calculation comes to a totally unoccluded view shown in Fig. 16b.

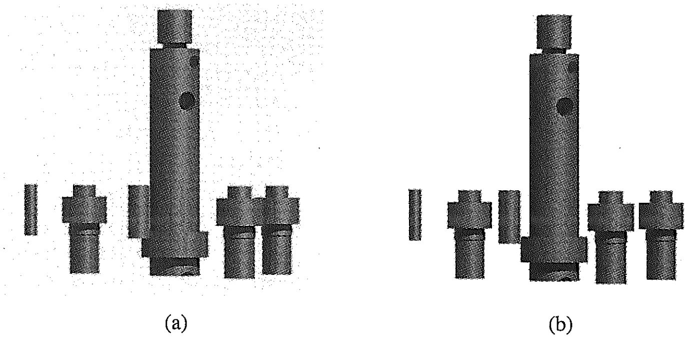


FIGURE 16 (a) Viewing the shaft and the pins, using the result from the SVD algorithm. (b) Viewing the shaft and the pins, using a totally unoccluded view direction.

2. The Visibility Function ($\mathcal{V}$)

A visibility function $\mathcal{V}$ is created to evaluate the quality of a camera view. The function is based on the visibility information of the assembly components of interest and some of the information already built into the assembly CAD structure. It is intuitive to desire the visibility of the components of interest. Also, the more numerous and the larger the faces visible from these components are, the easier the inspection task should be. As a result, the issue of visibility is addressed in terms of the number of visible components, the number of visible faces on each component, and the number of image pixels associated with each face.

$$\mathcal{V} = c_1 N_c + c_2 \sum_{i=1}^{N_c}(1-\exp^{-F_iF_i^0}) + c_3 \sum_{i=1}^{N_c} \frac{\sum_{j=1}^{F_i}(1-\exp^{-P_{ij}P_{ij}^0})}{F_i}, \tag{23}$$

where N_c is the number of visible components of interest; F_i is the number of visible faces on component i; F_i^0 is the distribution of the surface normals of the visible faces; P_{ij} is the number of pixels associated with face j of component i; P_{ij}^0 is the importance of face j; and c_1, c_2, c_3 are the constant coefficients specifying the contribution of each term.

The exponential function $(1-\exp^{-F_iF_i^0})$ shown in Fig. 17 is used so that the first few visible faces of a component increase $\mathcal{V}$ more than any additional faces; i.e., less increase is noticed as more faces of the same component become visible. As a result, among views with an equal number of visible faces (of components

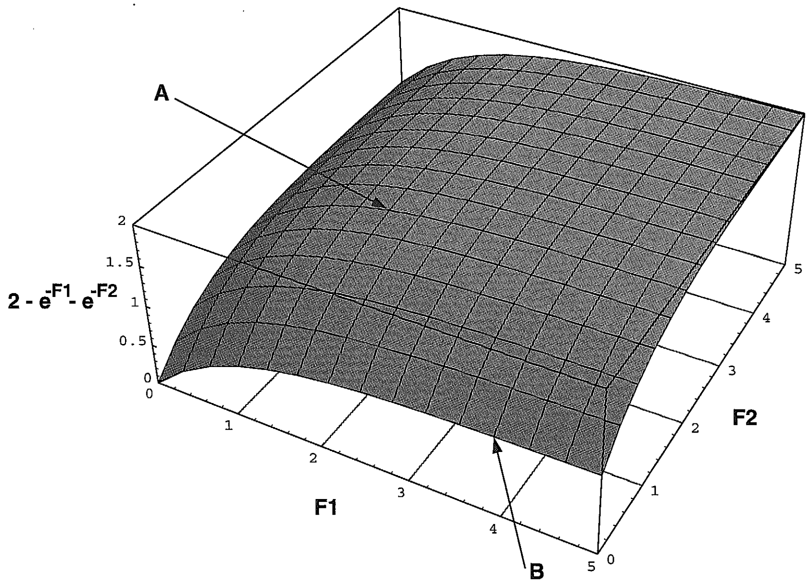


FIGURE 17 An exponential function is used to determine the contribution of visible faces from an assembly component of interest to the visibility function $\mathcal{V}$. Less increase is noticed as more faces become visible. The figure shows the case for two components of interest with $F_i^0 = 1$. Views with more equal distribution of visible faces over the components of interest are preferred. For example, note that point A, which has two faces visible from each component, is preferred over point B, which has four faces visible from only one component.

having equal values of F_i^0), the view with the most equal distribution of these faces among the components of interest generates the largest value of $\mathcal{V}$ (see Fig. 17). Similarly, the function $(1 - \exp^{-P_{ij}P_{ij}^0})$ is used to evaluate the contribution of the image pixels that are associated with the different visible faces.

The value of F_i^0 controls the rate of increase in the value of the exponential function with respect to F_i. Lower values of F_i^0 lead to lower rates of increase. It is known that faces whose surface normals span a wide range of orientations result in wider shading variations. As a result, these faces are easier to find in an image, and an error in the component of these faces is easier to detect. Therefore, F_i^0 is a measure of the variability of the surface normals of the visible faces on component i. It is calculated with the use of the singular values of the SVD of a matrix N_i that contains information about the orientation of the visible faces on component i. If a visible face j is planar with a normal unit vector $\hat{n}_j$, then $\hat{n}_j$ is placed in N_i to represent face j. On the other hand, if face j is nonplanar then the dominant orientations of the face are used to represent it in N_i. This is done by using the Σ and V matrices resulting from an SVD performed on a matrix S_j that contains the unit normals of all of the facets composing the face. So, if face j comprises k facets, then

$$S_j = \left[\hat{n}_1\, \hat{n}_2\, \cdots\, \hat{n}_k\right]^T = U\Sigma V^T \tag{24}$$

is performed, where $S_j \in R^{k\times 3}$. The matrices Σ and V are given by

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix} \in R^{k\times 3} \tag{25}$$

with $\sigma_1 \geq \sigma_2 \geq \sigma_3$ and

$$V = \left[\hat{v}_1 \quad \hat{v}_2 \quad \hat{v}_3\right] \in R^{3\times 3}. \tag{26}$$

Then face j is represented in N_i by the three vectors in V scaled such that their effect on the singular values of N_i is close to the effect of a single vector, i.e.,

$$\hat{v}_j^i = \frac{\sigma_i}{\sqrt{\sum_{l=1}^{3} \sigma_l^2}} \hat{v}_i. \tag{27}$$

Now, the SVD of N_i is performed:

$$N_i = U_i \Sigma_i V_i^T \tag{28}$$

where

$$\Sigma_i = \begin{bmatrix} \sigma_{i1} & 0 & 0 \\ 0 & \sigma_{i2} & 0 \\ 0 & 0 & \sigma_{i3} \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix} \in R^{(F_i^p+3F_i^c)\times 3}. \tag{29}$$

F_i^p is the number of the planar faces, F_i^c is the number of nonplanar curved surfaces, and $\sigma_{i1} \geq \sigma_{i2} \geq \sigma_{i3}$. The quantity F_i^0 is then calculated from

$$F_i^0 = \frac{\sigma_{i1} + \sigma_{i2} + \sigma_{i3}}{3\sigma_{i1}}, \tag{30}$$

where it is clear that $\frac{1}{3} < F_i^0 < 1$.

In an analogous manner, the coefficient P_{ij}^0 is used to emphasize faces that provide more information to the inspection algorithm. In this case, the displacement of faces that are in contact with other components is more likely to result in visible effects from assembly errors. Therefore, the value of P_{ij}^0 is made dependent on the contact information of face j of component i. Currently, it is set to a constant 0.1 if the face has any contacts and to a constant 0.05 if it does not. This contact information is readily available from the generated CAD solid model.

C. Light Placement

The effect of lighting conditions on the performance of the inspection algorithm was studied to design optimal lighting conditions for the assembly inspection work cell. This was done by performing simulations and then analyzing the algorithm's reaction to different lighting conditions. The results were used to develop an algorithm that attempts to solve the light source placement problem over a discretized range of camera locations. A metric $\mathscr{L}$ is devised to evaluate the quality of the solution at each camera location. The algorithm also considers the possibility of using multiple light sources.

1. Effect of Light Position on Performance

An experiment based on raytraced synthetic images was used to study the effect of light on the performance of the inspection algorithm. A simple pin-in-hole assembly, shown in Fig. 18, was used as a test assembly. A camera was simulated 45° from the top of the pin in the X-Y plane so that both the top face and portions of the side face are visible. Then different light positions were tested at intervals of 10°, also in the X-Y plane. At each position the inspection algorithm was trained on the correct assembly and then used to test assemblies with predetermined rotational and horizontal errors. The log likelihood value, described in Section II.B, that resulted from these tests is plotted as a function of light position to determine the effectiveness of the algorithm at detecting different types of errors.

First, the pin was rotated between −20° and 20° around the X axis centered at the center of the top surface of the pin. The samples in that region are 4° apart. Figure 19a shows a quadratic fit among the resulting points. Second, the pin is

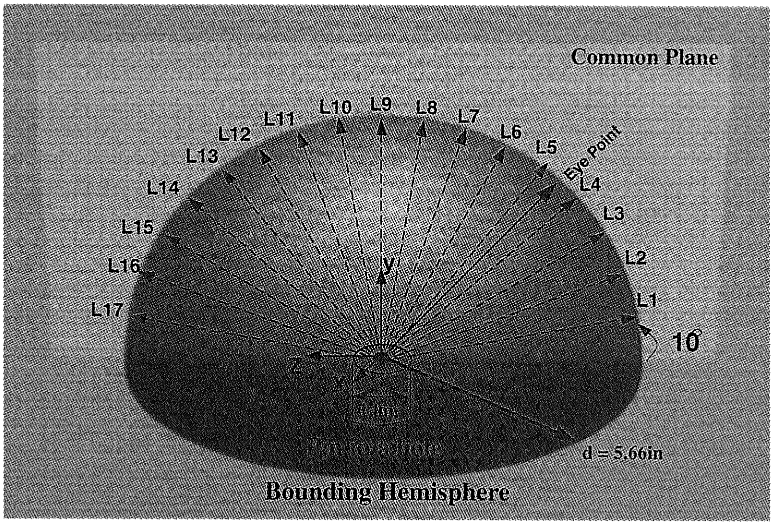


FIGURE 18 Experimental setup used to test the effect of light on the performance of the inspection algorithm. At every light position synthetic images are used to train the algorithm. Then errors are introduced to the images. The effectiveness of detecting these errors shows the effect of light on performance.

rotated between 0° and 20° around the Z axis (negative rotations around the Z axis would generate symmetrical images). The samples are also 4° apart. Figure 19b shows a quadratic fit among the resulting points. Finally, Fig. 19c shows the results from horizontal insertion errors. Note that in all three cases the algorithm is most sensitive to the errors when the light is around 135°, the perfect specular direction.

2. Light Source Placement Algorithm

The perfect specular direction was found to be most effective in the above experiment. A detailed analysis of the results from the above experiment leads to the conclusion that, in general, an assembly error is detected when it does one of the following in the inspection area:

1. Displaces a visible face such that its intensity due to specular reflection is increased or decreased
2. Uncovers or covers a face at an intensity different from that of the surrounding faces
3. Casts or removes a shadow, causing a different intensity.

An example illustrating this characterization is shown in Fig. 20. Part a of the figure shows the slice from Fig. 19a at light position 130°. The image of the pin correctly inserted in its hole is shown in Fig. 20b, and the image with −16° rotational error is shown in Fig. 20d. Fig. 20c plots the contribution of every pixel in the match area of Fig. 20b to the overall log likelihood match ratio. The darker the gray scale color of the pixel, the more it contributes to a mismatch. Similarly, Fig. 20e plots the contribution of every pixel in the match area of Fig. 20d to the overall log likelihood match ratio. It is clear that areas pointed to by arrows a and b caused the most mismatch. In area a the pin's side face disappeared, causing the

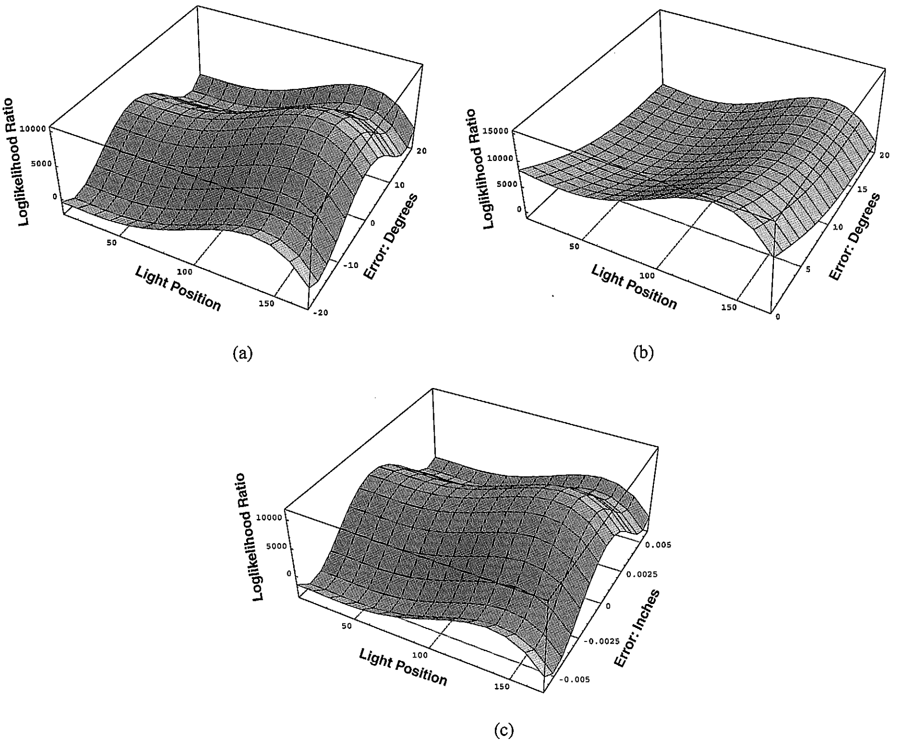
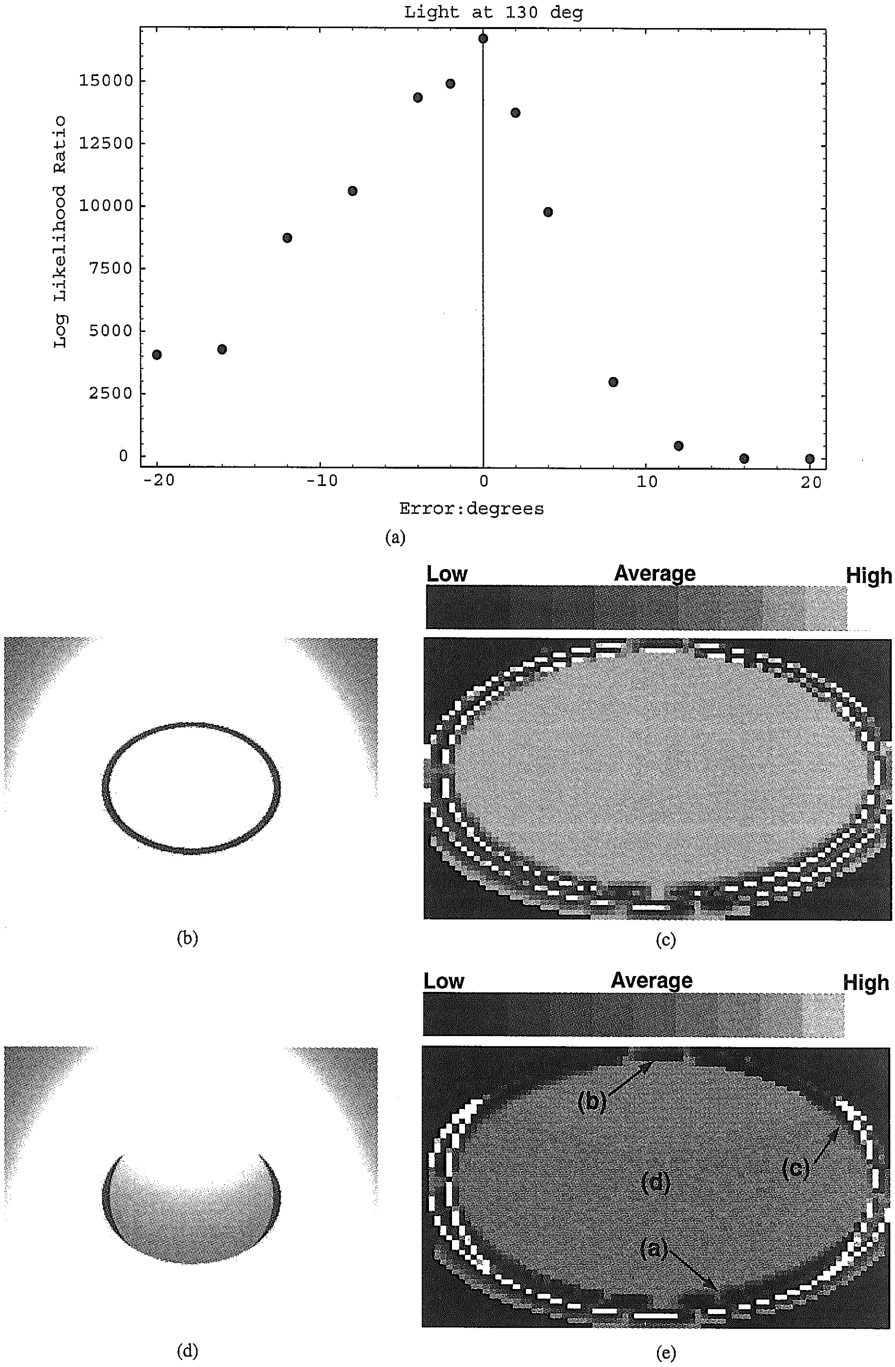


FIGURE 19 (a) A quadratic fit among the log likelihood match results from the experiment shown in Fig. 18 with rotational errors around the X axis. (b) Same as a, with rotational errors around the Z axis. (c) A quadratic fit among the log likelihood match results from the experiment shown in Fig. 18 with horizontal insertion errors.

change in gray-scale values to proceed from dark to light instead of light to dark to light again. This causes a mismatch between the Haar wavelet transform of this image and the error-free image. Similarly, the mismatch in area b was caused by the hole's side face disappearing. The fact that the pin's top face has a gray-scale shade different from the error-free case caused a different gray-scale value change at the side edges, which caused some mismatch, illustrated by the black band of pixels in area c. Moreover, this variation in the intensity of the top face resulted in a poor match, which is identified by dark gray-scale values in area d (compare this to the gray-scale values in Fig. 20c). It is important to remember that this match is obtained from the inspection algorithm that performs this task based on the statistical model it created during the training stage.

To determine an optimal light source location, it is clear that targeting the specular direction of the visible faces would enhance the first error detection mechanism. In addition, because the specular direction of a face covers a small subset of 3D space, the probability of uncovering (covering) a face at its specular shade is low. So, the probability of uncovering (covering) a face at an intensity different from that of the surrounding faces is high if the other faces have a high degree of intensity from specular reflection. As a result, an algorithm for placing a point light source is designed to target the specular direction of the visible faces. Because the



(a)

(b)

(c)

(d)

(e)

FIGURE 20 Analysis of the light experiment. (a) A slice from Fig. 19a at a 130° light position. (b) An error-free image of the pin in its hole. (c) A plot of the contribution of every pixel in the match area in b to the log likelihood ratio. Darker pixels contribute less. The black corners contain background information that is masked out by the algorithm and does not contribute to the match. (d) A −16° pin rotational error. (e) A plot of the contribution of every pixel in the match area in d to the log likelihood ratio.

specular direction of different visible faces will usually cover multiple directions, a least-squares solution is used to find an optimal light direction. The point light source is then placed on the bounding hemisphere along that direction.

To accomplish the above task, a pixel from each visible face is selected to be mapped back to the world coordinate system. If p^w_{ij} is a point mapped back from face j on component i, then view vector $\widehat{V}_{ij} = E - p^w_{ij}$ is used to calculate a vector $\hat{h}_{ij}$ that bisects the angle between $\widehat{V}_{ij}$ and $\widehat{N}_{ij}$, where $\widehat{N}_{ij}$ is the face's normal. A matrix $\boldsymbol{H}$ that consists of the $\hat{h}_{ij}$ vectors of all of the visible faces is created, and its SVD is used to find the dominant direction $\hat{h}$.

Multiple pixels can be selected to be mapped back from each planar face. In this case the vector $\hat{h}_{ij}$ that represents face j of component i in the $\boldsymbol{H}$ matrix is calculated by performing an SVD on a matrix $\boldsymbol{H}_{ij}$ of the $\hat{h}_{ijk}$ vectors calculated from each of the k pixels on face j and using the identified best fit. Furthermore, a curved face is represented by one $\hat{h}_{ij}$ vector by including all of its facets in creating the $\boldsymbol{H}_{ij}$ matrix. Using multiple points from each face leads to more accurate calculations, as will be shown later.

A point p^w_c that is the average of all p^w_{ijk} points is used to calculate the view vector $E - p^w_c$. If the angle between h and the view vector $E - p^w_c$ is θ, then the desired light direction $\widehat{R}$ is obtained through a clockwise rotation of $\hat{h}$ around $(E - p^w_c) \times \hat{h}$ by 3θ. Then, the light source location L is obtained by

$$L = c\widehat{R} + p^w_c, \tag{31}$$

where c is such that L lies on the bounding hemisphere.

Note that the h vectors are used instead of the specular direction of the visible faces because the faces' normals $\widehat{N}_{ij}$ are within 90° of $E - C$. So, $\hat{h}_{ij}$ is guaranteed to lie in the half-space containing the view vector. This constrains the result of the SVD to a half-space instead of a full space, had the specular direction of the visible faces been used directly.

3. Illumination Function ($\mathscr{L}$)

A function $\mathscr{L}$,

$$\mathscr{L} = \sum_{i=1}^{N_c}\left(\frac{F^v_i}{F_i}\left(\frac{\sum_{j=1}^{F_i}\sum_{k=1}^{n_{ij}}(L - p^w_{ijk})\cdot\widehat{R}_{ijk}}{2F_i n_{ij}} + 0.5\right)\right), \tag{32}$$

is calculated to evaluate the quality of the light source location, where N_c is the number of visible components of interest; F_i is the number of visible faces on component i; F^v_i is the number of F_i faces visible to the light; n_{ij} is the number of p^w_{ijk} points used from face j of component i; and $\widehat{R}_{kij}$ is the perfect specular direction for point p^w_{ijk}.

It measures the effectiveness of the calculated light position by attempting to measure the portion of the visible faces of an assembly component that is not shadowed from the light and to measure how close the light direction is to the perfect specular direction of these faces. Visibility of the p^w_{ijk} points to the light source is measured by using the Z-buffer hardware in a manner similar to the description in Section III.A.1, with the eye point placed at the light position's location. Clearly, mapping more visible pixels back to the world coordinate system leads to a more accurate measure; however, it also requires more computation time.

4. Multiple Light Sources with Image Fusion

The algorithm for finding a light source position described in Section III.C.2 targeted the specular direction of the visible faces. As $\mathcal{L}$ decreases because of a poorer fit to a wider range of different specular directions, the need for additional light sources increases. Duplicating the training and testing process is unreasonable because of the increased cost. In this section it is shown that the same number of images could be used for training and testing, although multiple images are taken of the assembly, one for every light source. This is done with the results from image fusion [33].

The experiment on the pin assembly that was described in Section III.C.1 is revised here to show the improvement accomplished by the use of image fusion. Training images of two different light source positions are fused together with the use of a simple pixel averaging scheme. After the inspection algorithm is trained on these images, inspection images are created with the kind of errors shown in Section III.C.1. However, for each error case, two images are created, each with one light turned on, and then fused together before being introduced to the inspection algorithm. Results have shown improvement in performance. For example, Fig. 21 shows the fusion of the training images of the light at 100° and 140°. Inspection images are generated in a similar fashion. Figure 22 shows the improvement gained from using the fusion method.

D. Generate-and-Test Analysis

To find an optimal camera–light pair, the region that the camera is constrained to is first discretized (see Fig. 23). At each discrete camera location, a point light source position is calculated with the algorithm described in Section III.C.2. A metric $\mathcal{M}$ given by

$$\mathcal{M} = C_{\mathcal{V}}\mathcal{V} + C_{\mathcal{L}}\mathcal{L}, \tag{33}$$

where $C_{\mathcal{V}}$ and $C_{\mathcal{L}}$ are constant factors, is used to evaluate the effectiveness of the camera and light source positions. The camera–light pair with the largest value of $\mathcal{M}$ is chosen for the assembly inspection task.

IV. RESULTS

The camera and light placement algorithm was initially tested on simple assemblies to verify its performance. Then it was tested on more complex assemblies. In this section the results from running the algorithm on the wheel assembly shown in Fig. 11 are presented.

The pins of the wheel assembly were used as the components of interest. The camera was constrained to lie on a semicircle as described in Section III.B.1 Then camera samples were evaluated 10° apart, and the different components of $\mathcal{M}$ were evaluated at every sample. Figure 24 plots the different components of $\mathcal{M}$. Curve a shows the second term of $\mathcal{V}$ (Eq. 23). It shows that the visibility of the components' faces diminishes at horizontal and vertical views. On the other hand, plot b, the third term of $\mathcal{V}$, shows higher values at these views because equally

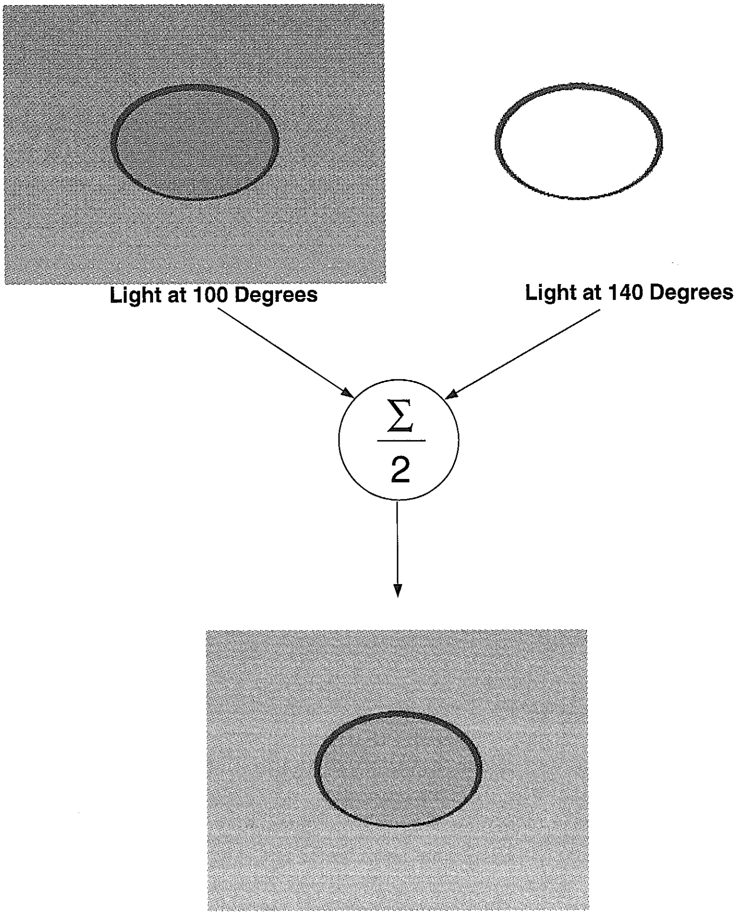


FIGURE 21 The figure shows two training images of the same pin assembly. One is from a light source at 100° (refer to Fig. 18), and the other is from a light source at a 140°. The two images are fused into one image with the use of a simple averaging method. The fused image is used for training.

large face areas are visible on the different assembly components of interest. Similarly, curve c of the $\mathscr{L}$ function shows better light positions at horizontal views because of the better match to the specular direction of the different visible faces. The combination of these terms in $\mathscr{M}$ leads to a preference for the inclined views. For example, setting all of the constants to unity (except for $c3$, which is set to 0.1) leads to selecting the view at 140° shown in Fig. 12. Testing the real assembly from different views after training on synthetic images showed the advantages of using the inclined views around 140°. Figure 25 shows two examples. Figure 25a shows a detected error caused by misplacing the top wheel. This error passes undetected from a horizontal view. Figure 25b shows a detected pin insertion error that passes undetected from a vertical view.

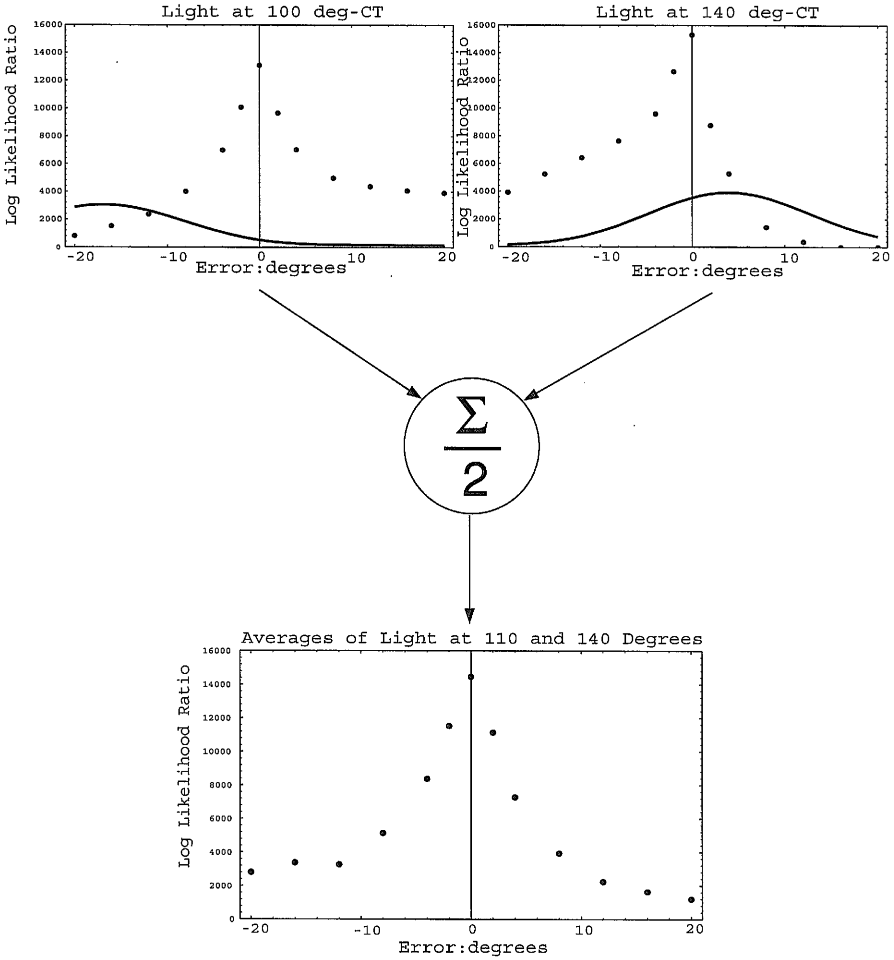


FIGURE 22 At the top, the figure shows the results at 100° and 140° from graph a in Fig 19. The dots represent the resulting match with respect to the different errors. The accompanying curve shows the calculated intensity from the Cook–Torrance model, which further illustrates the sensitivity of the algorithm to the specular direction. At the bottom, the figure shows the results after fusion of the two cases. Note that a smoother drop on both sides of the curve is accomplished.

V. CONCLUSIONS

This chapter has discussed an intelligent assembly inspection system that uses a multiscale algorithm to detect errors in assemblies after the algorithm is trained on synthetic CAD images of correctly assembled products. It was shown how the CAD information of an assembly along with fast rendering techniques on specialized graphics machines can be used for the automation of the work-cell camera and light placement. The current emphasis in the manufacturing industry on concurrent engineering will only cause this integration between the CAD model of products and its manufacturing inspection to grow in value.

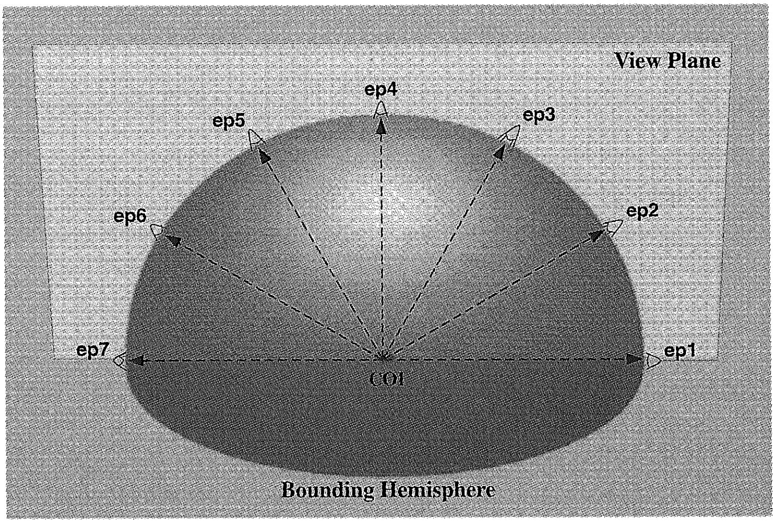


FIGURE 23 The figure shows the assembly's bounding hemisphere that the camera is constrained to lie on. The algorithm described in Section III.B.1 creates the SVD plane as an additional constraint. The resulting region is sampled at some interval (the figure shows 30°). A metric $\mathcal{M}$ is evaluated at each sample to select the best point possible.

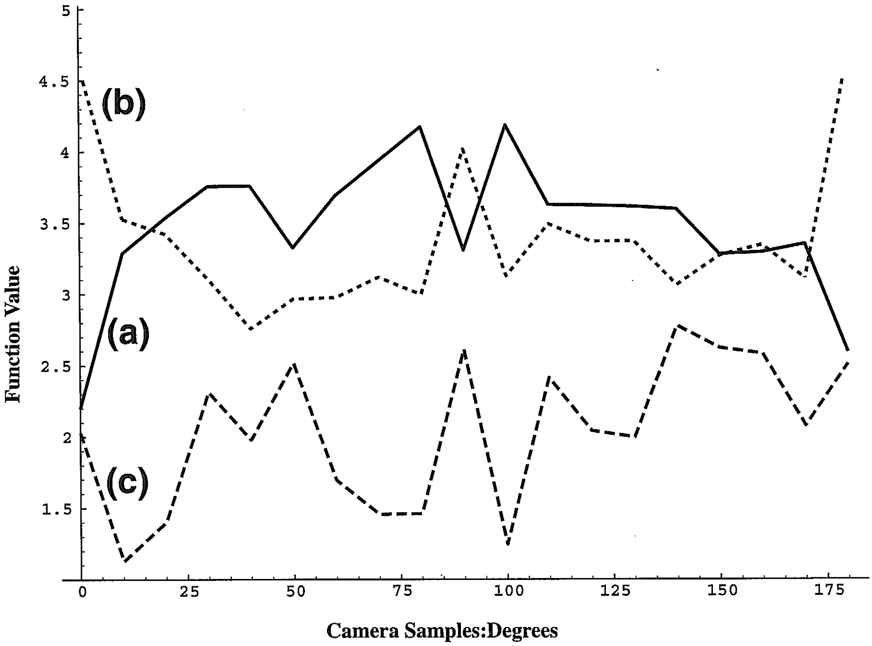


FIGURE 24 This figure plots the value of the components of $\mathcal{M}$, viewing the wheel assembly shown in Fig. 12 at the different camera samples on the constraining semicircle. The pins in the assembly were used as the components of interest. The samples are 10° apart. The first term of $\mathcal{V}$ is ignored because it is constant at one. (a) The term associated with the number of faces visible on every component (the second term of $\mathcal{V}$). (b) The term related to the size of the faces' visible areas on the components (the third term of $\mathcal{V}$). (c) The $\mathcal{L}$ function.

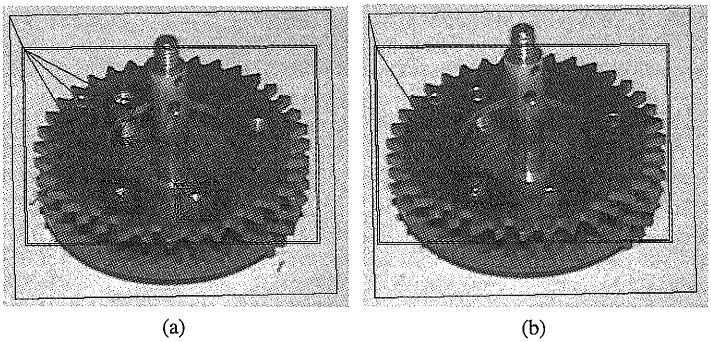


FIGURE 25 (a) Error in top wheel placement. The location of a mismatch is identified by a rectangle with an X mark. The tree shown in Fig. 12 is used here. (b) Error in high-density pin insertion.

REFERENCES

1. Tarabanic, K. A., Allen, P. K., and Tsai, R. Y. "A Survey of Sensor Planning in Computer Vision." *IEEE Trans. Robot. Automat.* 11(1):86–104, 1995.
2. Tarabanis, K., Tsai, R. Y., and Allen, P. K. Automated sensor planning for robotic vision tasks. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, Sacramento, CA, April, 1991, pp. 76–82.
3. Cowan, C. K. Automatic camera and light-source placement using CAD models. In *Proceedings of the IEEE Workshop on Directions in Automated CAD-Based Vision*, Maui, HI, June 2–3, 1991, pp. 22–31.
4. Yi, S., Haralick, R. M., and Shapiro, L. G. Automatic sensor and light source positioning for machine vision. In *Proceedings of the 10th International Conference on Pattern Recognition*, 1990, pp. 55–59.
5. Sakane, S., Ishii, M., and Kakikura, M. "Occlusion Avoidance of Visual Sensors Based on a Hand-Eye Action Simulator System: HEAVEN." *Adv. Robot.* 2(2):149–165, 1987.
6. Sakane, S. and Sato, T. Automatic planning of light source and camera placement for an active photometric stereo system. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, Sacramento, CA, April 1991, pp. 1080–1087.
7. Solomon, F. and Ikeuchi, K. An illumination planner for Lambertian polyhedral objects. In *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, Nagoya, Aichi, Japan, May 21–27, 1995, pp. 1719–1725.
8. Chellappa, R. and Kashyap, R. "Digital Image Restoration Using Spatial Interaction Models." *IEEE Trans. Acoust. Speech Signal Processing* 30(3):461–471, 1982.
9. Besag, J. "On the Statistical Analysis of Dirty Pictures." *J. R. Statist. Soc. B* 48(3):259–302, 1986.
10. Bouman, C. and Shapiro, M. "A Multiscale Random Field Model For Bayesian Image Segmentation." *IEEE Trans. Image Processing* 3(2):162–177, 1994.
11. Benveniste, A., Nikoukhah, R., and Willsky, A. Multiscale system theory. In *Proceedings of the 29th Conference on Decision and Control*, December 1990, pp. 2484–2489.
12. Bassevile, M., Benveniste, A., Chou, K. C., Golden, S. A., Nikoukhah, R., and Willsky, A. S. "Modeling and Estimation of Multiresolution Stochastic Processes." *IEEE Trans. Inform. Theory* 38(2): 766–784, 1992.
13. Amit, Y., Grenander, U., and Piccioni, M. "Structural Image Restoration Through Deformable Templates." *J. Amer. Statist. Assoc.* 86(414):376–387, 1991.
14. Kam, A. and Kopec, G. Heuristic image decoding using separable source models. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, Adelaide, South Australia, April 19–22, 1994, pp. 145–148.
15. Kopec, G. and Chou, P. "Document Image Decoding Using Marcov Source Models." *IEEE Trans. Pattern Anal. Machine Intell.* 16(6):602–617, 1994.
16. Chin, R. and Harlow, C. "Automated Visual Inspection: A Survey." *IEEE Trans. Pattern Anal. Machine Intell.* 4(6):557–573, 1982.

17. Brooks, R. "Mode-Based Three-Dimensional Interpretations of Two-Dimensional Images." *IEEE Trans. Pattern Anal. Machine Intell.* 5(2):140–150, 1983.
18. Flynn, P. and Jain, A. "CAD-Based Computer Vision: From CAD Models to Relational Graphs." *IEEE Trans. Pattern Anal. Machine Intell.* 13(1):114–132, 1991.
19. Mehrotra, R. and Grosky, W. "Shape Matching Utilizing Indexed Hypotheses Generation and Testing." *IEEE Trans. Robotics Automat.* 5(1):70–77, 1989.
20. Gunnarsson, K. and Prinz, F. "CAD Model-Based Localization of Parts in Manufacturing." *Computer* 20(8):66–74, 1987.
21. Shariat, H. A model-based method for object recognition. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, Cincinnati, OH, May 13–18, 1990.
22. Tretter, D., Bouman, C. A., Khawaja, K. W., and Maciejewski, A. A. "A Multiscale Stochastic Image Model for Automated Inspection." *IEEE Trans. Image Processing* 4(12):1641–1654, 1995.
23. Kirćanski, M. V. and Borić, M. D. "Symbolic Singular Value Decomposition for a PUMA Robot and its Applications to a Robot Operation near Singularities." *Int. J. Robot. Res.* 12(5):460–472, 1993.
24. Dempster, A. P., Laird, N. M., and Rubin, D. B. "Maximum Likelihood from Incomplete Data Via the EM Algorithm." *J. R. Statist. Soc. B* 39(1):1–38, 1977.
25. Pearl, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.
26. Mallat, S. "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation." *IEEE Trans. Pattern Anal. Machine Intell.* 11(7):674–693, 1989.
27. Burt, P. Smart sensing within a pyramid vision machine. *Proc. IEEE* 76(8):1006–1015, 1988.
28. Clarke, R. *Transform Coding of Images*. Academic Press, Orlando, FL, 1985.
29. Khawaja, K. W., Tretter, D., Maciejewski, A. A., and Bouman, C. A. Automated assembly inspection using a multiscale algorithm trained on synthetic images. In *Proceedings of the 1994 International Conference on Robotics and Automation*, San Diego, CA, May 8–13, 1994, pp. 3530–3536.
30. Watt, A. and Watt, M. *Advanced Animation and Rendering Techniques*. Addison-Wesley, New York, 1992.
31. Whitted, T. "An Improved Illumination Model for Shaded Display." *Commun. ACM* 23(6):343–349, 1980.
32. Cook, R. L. and Torrance, K. E. "A Reflectance Model for Computer Graphics." *ACM Trans. Graph.* 1(1):7–24, 1982.
33. Li, H., Manjunath, B. S., and Mitra, S. K. Multisensor image fusion using the wavelet transform. *Graphical Models Image Processing* 57(3):235–245, 1995.