# 8 Robot programming and robot vision

So far, we have discussed in some detail the automatic processing and analysis of images and image data. Any information about position and orientation of objects has been derived in an image frame for reference, that is, with respect to the two-dimensional array of pixels representing the imaged scene. However, objects only exist in the so-called real world (i.e. the world we are imaging) and if we wish to do something useful with the information we have extracted from our digital images, then we have to address the relationship between images and the three-dimensional environment from which they are derived. Even then, this is typically not enough since there is little point in knowing where things are if we don't know how to manipulate and move them. In this chapter, we turn our attention to these issues and we will discuss how we can usefully describe the three-dimensional position and orientation of objects relative to any other object, how to go about configuring a task to effect their manipulation, and how, given their image position and orientation, we can derive their three-dimensional pose.

Recall that imaging is a projective process from a three-dimensional world to a two-dimensional one and, as such, we lose one dimension (typically the object depth or range) when we perform this imaging. As we noted in Chapter 1, much of computer vision is concerned with the recovery of this third dimension but, so far in this book, we have not dealt with it in any detail. Most of the discussion of recovery of depth information is included in the next chapter on image understanding. However, to make this chapter complete and self-contained, we discuss at the end a popular and versatile technique called *structured-light*, which allows us to compute the distance between a camera and objects in the field of view.

Before proceeding, we will briefly review robot programming methodologies to provide a context for the approach that we adopt in this book.

## 8.1 A brief review of robot programming methodologies

Modern robotic manipulators, which have their origins in telecheric and numerically controlled devices, have been developing for the past twenty-five years. As robots have developed, so too have various methods evolved for programming them and, consequently, modern commercially available robot manipulators make use of many programming techniques which exhibit a wide spectrum of sophistication. There are, broadly speaking, three main categories of robot programming system which are, in order of the level of sophistication: guiding systems; robot-level or explicit-level systems; and task-level systems.

Guiding systems are typified by the manual lead-through approach in which the manipulator is trained by guiding the arm through the appropriate positions using, for example, a teach-pendant, and recording the individual joint positions. Task execution is effected by driving the joints to these recorded positions. This type of manual teaching is the most common of all programming systems, and though several variations on the theme have been developed, such as trajectory control, the explicit interactive nature remains the same.

Robot-level programming systems, for the most part, simply replace the teach-pendant with a robot programming language: manipulator movements are still programmed by explicitly specifying joint positions. However, several languages also facilitate robot control in a three-dimensional Cartesian space, rather than in the joint space. This is effected using the *inverse kinematic solution* of the manipulator arm (the kinematic solution allows you to compute the position of the end-effector or gripper in a three-dimensional Cartesian frame of reference, given the manipulator joint positions; the inverse kinematic solution allows you to compute the joint positions for a given position and orientation of the end-effector). The more advanced of these languages incorporate structured programming control constructs (such as are found in most modern programming languages, e.g. ADA, Modula-2, and Pascal) and they make extensive use of *coordinate transformation* and *coordinate frames*. With this approach, the robot control is defined in terms of transformations on a coordinate frame (a set of $XYZ$-axes) associated with, and embedded in, the robot hand. Off-line programming is more feasible as long as the transformations representing the relationships between the frames describing the objects in the robot environment are accurate. It is this approach which we adopt in this book and these techniques will be discussed in detail in the remainder of the chapter.

Task-level robot programming languages attempt to describe assembly tasks as sequences of goal spatial relationships between objects and, thus, they differ from the other two approaches in that they focus on the objects rather than on the manipulator. The robot is merely a mechanism to achieve these goals. They typically require the use of task planning, path planning, collision avoidance and world-modelling: this level of sophistication is not yet widely available on a commercial basis.

## 8.2 Description of object pose with homogeneous transformations

Robot manipulation is concerned, in essence, with the spatial relationships between several objects, between objects and manipulators, and with the reorganization of these relationships. We will use *homogeneous transformations* to represent these spatial relationships. However, we first need to review a little vector algebra to ensure that we are equipped with the tools to develop this methodology.

A vector $v = ai + bj + ck$, where $i, j$ and $k$ are unit vectors along the $X$-, $Y$-, and $Z$-axes of a coordinate reference frame (see Figure 8.1), is represented in *homogeneous coordinates* as a column matrix:

$$v = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

where:

$$a = \frac{x}{w}$$

$$b = \frac{y}{w}$$

$$c = \frac{z}{w}$$

Thus, the additional fourth coordinate $w$ is just a scaling factor and means that a single three-dimensional vector can be represented by several homogeneous coordinates. For example, $3i + 4j + 5k$ can be represented by

$$\begin{bmatrix} 3 \\ 4 \\ 5 \\ 1 \end{bmatrix} \text{ or by } \begin{bmatrix} 6 \\ 8 \\ 10 \\ 2 \end{bmatrix}.$$

Note that, since division of zero by zero is indeterminate, the vector

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

is undefined.

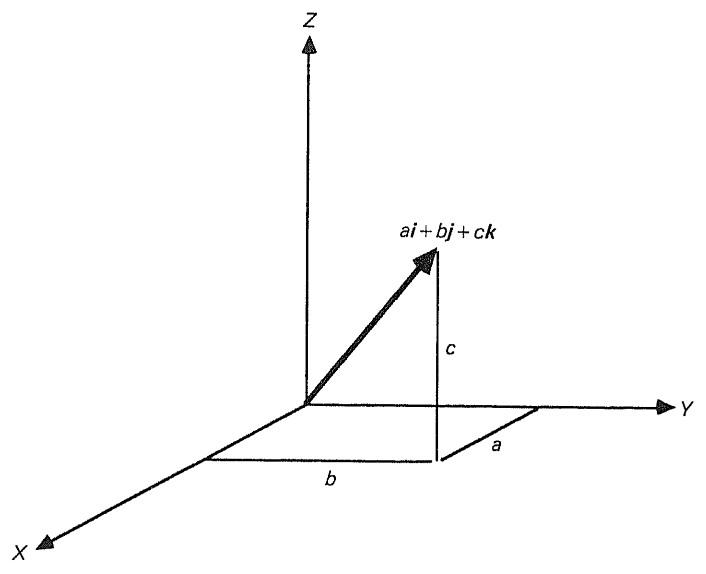


**Figure 8.1** Coordinate reference frame.

Vectors can be combined in two simple ways. Given two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$:

$$\boldsymbol{a} = a_x\boldsymbol{i} + a_y\boldsymbol{j} + a_z\boldsymbol{k}$$
$$\boldsymbol{b} = b_x\boldsymbol{i} + b_y\boldsymbol{j} + b_z\boldsymbol{k}$$

The *vector dot product* is defined:

$$\boldsymbol{a} \cdot \boldsymbol{b} = a_x b_x + a_y b_y + a_z b_z$$

and is a scalar quantity. The *vector cross product* is defined:

$$\boldsymbol{a} \times \boldsymbol{b} = (a_y b_z - a_z b_y)\boldsymbol{i} + (a_z b_x - a_x b_z)\boldsymbol{j} + (a_x b_y - a_y b_x)\boldsymbol{k}$$

It may also be written as:

$$\boldsymbol{a} \times \boldsymbol{b} = \begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{bmatrix}$$

that is, as the expansion of this $3 \times 3$ determinant.

A general transformation $H$, in three-dimensional space, representing translation, rotation, stretching, and perspective distortions, is a $4 \times 4$ matrix in homogeneous formulation. Given a point represented by the vector $\boldsymbol{u}$, its transformation $\boldsymbol{v}$ is represented by the matrix product:

$$\boldsymbol{v} = H\boldsymbol{u}$$

The transformation $H$ corresponding to a translation by a vector $ai + bj + ck$ is:

$$H = TRANS(a, b, c) = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For example: to transform $u = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$ by $H$:

$$v = Hu = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

$$= \begin{bmatrix} x + aw \\ y + bw \\ z + cw \\ w \end{bmatrix}$$

$$= \begin{bmatrix} x/w + a \\ y/w + b \\ z/w + c \\ 1 \end{bmatrix}$$

Thus, we have the familiar property of translation of a vector by another vector being simply the addition of their respective coefficients.

The transformation corresponding to rotations about the $X$-, $Y$-, or $Z$-axes by an angle $\theta$ are:

$$Rot(X, \theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot(Y, \theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot(Z, \theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now, from our point of view, we come to the most important aspect of homogeneous transformations in that we can interpret the homogeneous transformation as a *coordinate reference frame*. In particular, a homogeneous

transformation describes the position and orientation of a coordinate frame with respect to another previously defined coordinate frame. Thus, the homogeneous transformation represents not only transformations of vectors (points) but also positions and orientations.

Specifically, a coordinate frame is defined by four things: the position of its origin and the direction of its $X$-, $Y$-, and $Z$-axes. The first three columns of the homogeneous transformation represent the direction of the $X$-, $Y$-, and $Z$-axes of the coordinate frame with respect to the base coordinate reference frame, while the fourth column represents the position of the origin. This is intuitively appealing: a homogeneous transformation, which can be a combination of many simpler homogeneous transformations, applies equally to other homogeneous transformations as it does to vectors (pre-multiplying a $4 \times 1$ vector by a $4 \times 4$ matrix yields a $4 \times 1$ vector; pre-multiplying a $4 \times 4$ matrix by a $4 \times 4$ matrix yields a $4 \times 4$ matrix which is still a homogeneous transformation). Thus, we can take a coordinate reference frame and move it elsewhere by applying an appropriate homogeneous transformation. If the coordinate frame to be 'moved' is originally aligned with the so-called base coordinate reference frame, then we can see that the homogeneous transformation is both a description of how to transform the base coordinate frame to the new coordinate frame and a description of this new coordinate frame with respect to the base coordinate reference frame. For example, consider the following transformation:

$$H = Trans\ (10, 10, 0)\ Rot\ (Y, 90)$$

$$= Trans\ (10, 10, 0) \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 1 & 10 \\ 0 & 1 & 0 & 10 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The transformation of the three vectors corresponding to the unit vectors along the $X$-, $Y$-, and $Z$-axes are:

$$\begin{bmatrix} 0 & 0 & 1 & 10 \\ 0 & 1 & 0 & 10 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 10 \\ 10 \\ -1 \\ 1 \end{bmatrix},$$

$$\begin{bmatrix} 0 & 0 & 1 & 10 \\ 0 & 1 & 0 & 10 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 10 \\ 11 \\ 0 \\ 1 \end{bmatrix}, \text{ and}$$

$$\begin{bmatrix} 0 & 0 & 1 & 10 \\ 0 & 1 & 0 & 10 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 11 \\ 10 \\ 0 \\ 1 \end{bmatrix} \text{ respectively}$$

The direction of these (transformed) unit vectors is formed by subtracting the vector representing the origin of this coordinate frame and extending the vectors to infinity by reducing the scale factor to zero. Thus, the direction of the $X$-, $Y$-, and $Z$-axes of this (new) frame are

$$\begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \text{ and } \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Similarly, the transformation of the null vector, i.e. the vector which performs no translation and thus defines the origin of the base coordinate frame, is given by:

$$\begin{bmatrix} 10 \\ 10 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 10 \\ 0 & 1 & 0 & 10 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

These four results show us that the new origin is at coordinates (10, 10, 0); the new $X$-axis is directed along the $Z$-axis of the base coordinate reference frame in the negative direction; the new $Y$-axis is directed along the $Y$-axis of the base coordinate reference frame in the positive direction; and the new $Z$-axis is directed along the $X$-axis of the base coordinate reference frame in the positive direction. This can be seen in Figure 8.2. You should try to do this transformation graphically but remember when deciding in which sense to make a rotation that: a positive rotation about the $X$-axis takes the $Y$-axis *towards* the $Z$-axis; a positive rotation about the $Y$-axis takes the $Z$-axis *towards* the $X$-axis; a positive rotation about the $Z$-axis takes the $X$-axis *towards* the $Y$-axis.

The rotations and translations we have been describing have all been made relative to the fixed base reference frame. Thus, in the transformation given by:

$$H = Trans(10, 10, 0)\ Rot\ (Y, 90)$$

the frame is first rotated around the reference $Y$-axis by 90°, and *then* translated by $10i + 10j + 0k$.

This operation may also be interpreted in reverse order, from left to right, viz: the object (frame) is first translated by $10i + 10j + 0k$; it is then rotated by 90° around the *station* frame axis ($Y$). In this instance, the effect is the same, but in general it will not be. This second interpretation seems to be the most intuitive since we can forget about the base reference frame and just remember 'where we are': our current station coordinate reference frame. We then just need to decide what

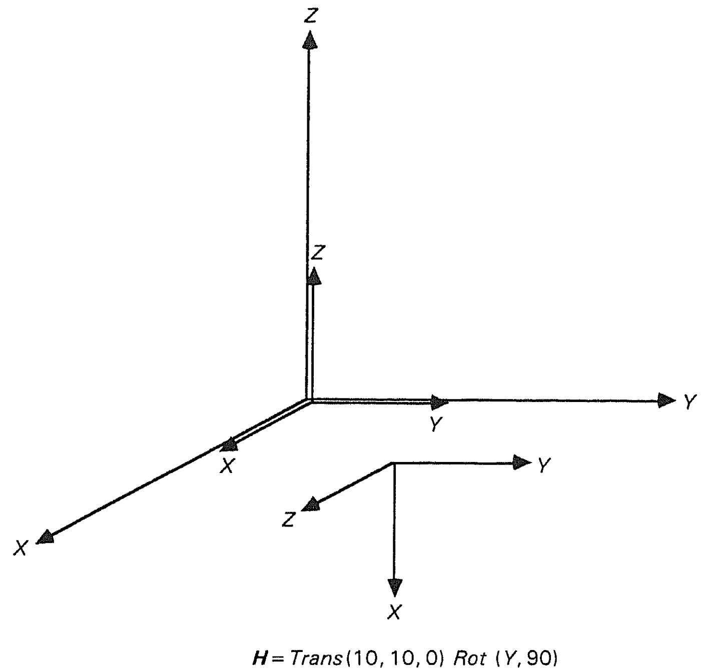


**Figure 8.2** Interpreting a homogeneous transformation as a coordinate frame.

transformations are necessary to get us to where we want to be based on the orientation of the station axes. In this way, we can get from pose to pose by incrementally identifying the appropriate station transformations, $H_1, H_2, H_3, \ldots H_n$, which we apply sequentially, as we go, and the final pose is defined with respect to the base simply as:

$$H = H_1 * H_2 * H_3 * \ldots * H_n.$$

In order to clarify the relative nature of these transformations, each of these frames/transformations is normally written with a leading superscript which identifies the coordinate frame with respect to which the (new) frame/transformation is defined. The leading superscript is omitted if the defining frame is the base frame. Thus the above transform equation is more correctly written:

$$H = H_1 * {}^{H_1}H_2 * {}^{H_2}H_3 * \ldots * {}^{H_{n-1}}H_n.$$

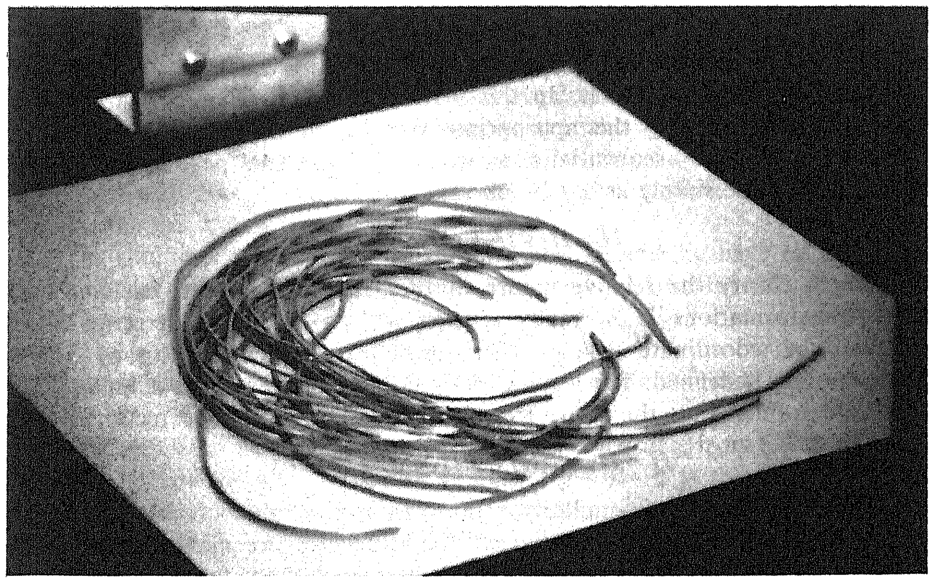
As a general rule, if we post-multiply a transform representing a frame by a second transformation describing a rotation and/or translation we make that rotation/transformation with respect to the frame axis described by the first transformation. On the other hand, if we pre-multiply the frame transformation representing a rotation/transformation then the rotation/transformation is made with respect to the base reference coordinate frame.

At this stage, we have developed a system where we can specify the position and orientation of coordinate reference frames anywhere with respect to each other and with respect to a given base frame. This, in itself, is quite useless since the world you and I know does not have too many coordinate reference frames in it. What we really require is a way of identifying the pose of *objects*. In fact, we are about there. The trick, and it is no more than a trick, is to *attach* a coordinate frame to an object, i.e. symbolically glue an $XYZ$-frame into an object simply by defining it to be there. Now, as we rotate and translate the coordinate frame, so too do we rotate and translate objects. We shall see this at work in the next section on robot programming.

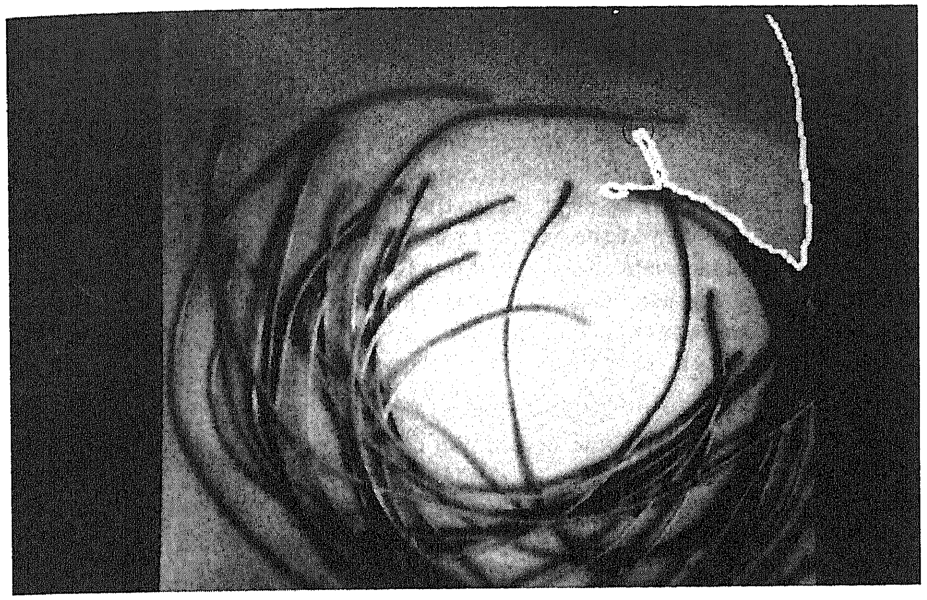
## 8.3 Robot programming: a wire crimping task specification

Perhaps the best way to introduce robot programming is by example and, so, we will develop our robot programming methodology in the context of a specific application: automated wire crimping.

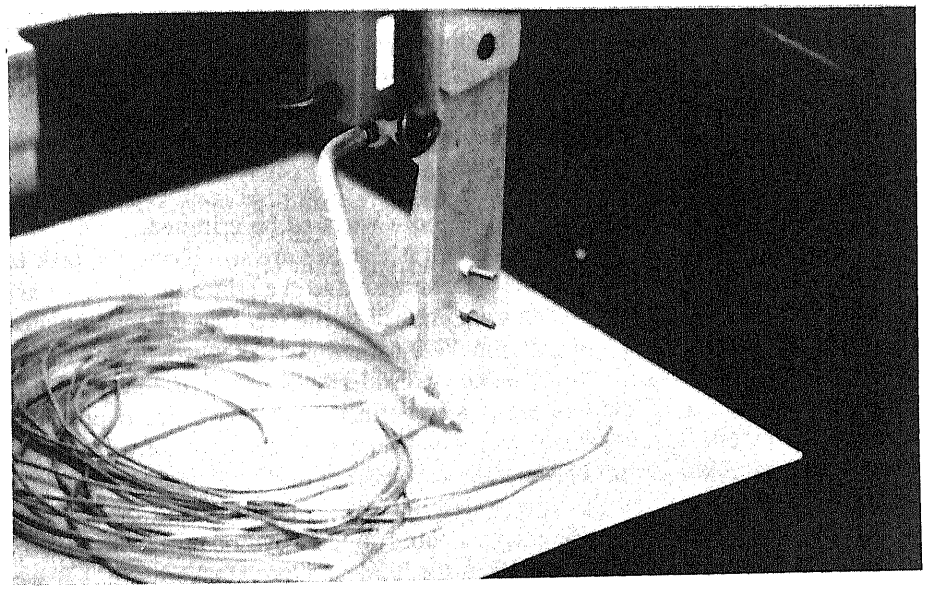
The wire crimping task requires that a robot manipulator grasp a wire from a tray of wires (see Figure 8.3). The wires are short and curved but they are flexible and the curvature varies from wire to wire. We will assume that, although the wires



(a)



(b)



(c)

**Figure 8.3** (a) A tray of flexible electrical wires (b) an image of which yields a point on a wire which is suitable for grasping (c) allowing the robot to pick up the wire and manipulate it.

overlap, they all lie flat on the tray. The wire must be grasped near its end by the robot, *at a point which has been identified using a vision system*, and the wire end must be inserted in a crimping machine, and the crimped end then must be inserted in a plastic connector. There are a few other related tasks, such as crimping the other end and inserting it in another connector, but, for the purposes of this example, we will assume that the task is complete once the wire has been crimped the first time. In addition, we must ensure that the manipulator does not obstruct the camera's field of view at the beginning of each crimp cycle and, thus, it must be moved to an appropriate remote position. By defining a series of manipulator end-effector positions *Mn*, say, this task can be described as a sequence of manipulator movements and actions referred to these defined positions. For example, the task might be formulated as follows:

*M*0: Move out of the field of view of the camera.
Determine the position and orientation of the wire-end and the grasp point using the vision system.
*M*1: Move to a position above the centre of the tray of wires.
*M*2: Move to an approach position above the grasp point.
*M*3: Move to the grasp position
Grasp the wire.
*M*4: Move to the depart position above the grasp point.
*M*5: Move to the approach position in front of the crimp.
*M*6: Move to a position in which the wire-end and the crimp are in contact.
*M*7: Move to a position such that the wire-end is inserted in the crimp.
Actuate the crimping machine.
*M*8: Move to the depart position in front of the crimping machine.
*M*9: Move to a position above the collection bin.
Release the wire.

This process is repeated until there are no more wires to be crimped.

One of the problems with this approach is that we are specifying the task in terms of movements of the robot while it is the wire and the crimp in which we are really interested. The object movements are implicit in the fact that the manipulator has grasped it. However, we will try to make up for this deficiency to some extent when we describe the structure of the task by considering the structure of the task's component objects: the manipulator, the end-effector, the wire grasp position, the wire end, the crimping machine, and the crimp. In particular, we will use the explicit positional relationships between these objects to describe the task structure. Since coordinate frames can be used to describe object position and orientation, and since we may need to describe a coordinate frame in two or more ways (there is more than one way to reach any given position and orientation), we will use transform equations to relate the two descriptions. A simple example, taken from *'Robot manipulators'* by R. Paul (1981), will serve to illustrate the approach.

Consider the situation, depicted in Figure 8.4, of a manipulator grasping a toy block. The coordinate frames which describe this situation are as follows:

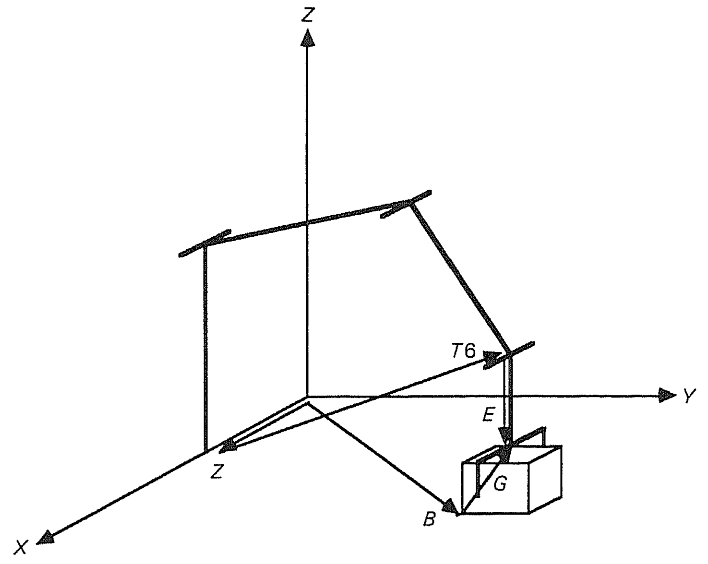


**Figure 8.4** A manipulator grasping a block.

| | |
|---|---|
| $Z$ | is the transform which describes the position of manipulator with respect to the base coordinate reference frame. |
| ${}^{Z}T6$ | describes the end of the manipulator (i.e. the wrist) with respect to the base of manipulator, i.e. with respect to $Z$. |
| ${}^{T6}E$ | describes the end-effector with respect to the end of the manipulator, i.e. with respect to $T6$. |
| $B$ | describes a block's position with respect to the base coordinate reference frame. |
| ${}^{B}G$ | describes the manipulator end-effector with respect to the block, i.e. with respect to $B$. |

In this example, the end-effector is described in two ways, by the transformations leading from the base to the wrist to the end-effector:

$$Z * {}^{Z}T6 * {}^{T6}E$$

and by the transformations leading from the block to the end-effector grip position:

$$B * {}^{B}G$$

Equating these descriptions, we get the following transform equation:

$$Z * {}^{Z}T6 * {}^{T6}E = B * {}^{B}G$$

Solving for $T6$ by multiplying across by the inverse of $Z$ and ${}^{T6}E$:

$${}^{Z}T6 = Z^{-1} * B * {}^{B}G * {}^{T6}E^{-1}$$

$T6$ is a function of the joint variables of the manipulator and, if known, then the appropriate joint variables can be computed using the inverse kinematic solution. $T6$, then, is the coordinate which we wish to program in order to effect the manipulation task: an arm position and orientation specified by $T6$ is thus equivalent to our previous informal movement $Mn$:

$$\text{Move } Mn = \text{Move } {}^{Z}T6$$

and, since we can compute $T6$ in terms of our known frames, we now have an arm movement which is specified in terms of the frames which describe the task structure. Assigning the appropriate value to $T6$ and moving to that position, implicitly using the inverse kinematic solution:

$$\begin{gathered} {}^{Z}T6 := Z^{-1} * B * {}^{B}G * {}^{T6}E^{-1} \\ \text{Move } {}^{Z}T6 \end{gathered}$$

What we have not yet done, and we will omit in this instance, is to fully specify each of these frames by embedding them in the appropriate objects and specifying the transformations which define them. We will do this in full for the wire crimping application.

Before proceeding, it is worth noting that, in this example, as the position of the end-effector with respect to the base reference system is represented by:

$$Z * {}^{Z}T6 * {}^{T6}E$$

this allows you to generate general-purpose and reusable robot programs. In particular, the calibration of the manipulator to the workstation is represented by $Z$, while if the task is to be performed with a change of tool, only $E$ need be altered.

Returning again to the wire crimping application, the transforms (i.e. frames) which are used in the task are as follows.

As before:

| | |
|---|---|
| $Z$ | is the transform which describes the position of manipulator with respect to the base coordinate reference frame. |
| ${}^{Z}T6$ | describes the end of the manipulator (i.e. the wrist) with respect to the base of manipulator, i.e. with respect to $Z$. |
| ${}^{T6}E$ | describes the end-effector with respect to the end of the manipulator, i.e. with respect to $T6$. |

We now define:

| | |
|---|---|
| $OOV$ | the position of the end-effector out of the field of view of the camera and defined with respect to the base coordinate reference system. |
| $CEN$ | the position of the end-effector centred over table defined with respect to the base coordinate reference system. |
| $WDUMP$ | the position of the end-effector over the bin of crimped wires, defined with respect to the base coordinate reference system. |

| | |
|---|---|
| $W$ | the position of the wire end, defined with respect to the base coordinate reference system. |
| ${}^{W}WG$ | the position of end-effector holding wire, defined with respect to the wire end. |
| ${}^{WG}WA$ | the position of end-effector approaching grasp position, defined with respect to the wire-grasp position. |
| ${}^{WG}WD$ | the position of end-effector departing grasp position (having grasped the wire), defined with respect to the original wire-grasp position. |
| $CM$ | the position of the crimping machine, defined with respect to the base coordinate reference system. |
| ${}^{CM}C$ | the position of the crimp (ready to be attached), defined with respect to the crimping machine. |
| ${}^{C}CA$ | the position of the wire end approaching crimp, defined with respect to the crimp. |
| ${}^{C}CC$ | the position of the wire end in contact with the crimp, defined with respect to the crimp. |
| ${}^{C}CI$ | the position of the wire end inserted in the crimp, defined with respect to the crimp. |
| ${}^{C}CD$ | the position of the wire end departing from the crimping machine (the crimp having been attached), defined with respect to the crimp. |

The manipulator movements $M0$ through $M9$ can now be expressed as combinations of these transforms:

$$\begin{aligned}
M0&: \quad T6 = Z^{-1} * OOV * E^{-1} \\
M1&: \quad T6 = Z^{-1} * CEN * E^{-1} \\
M2&: \quad T6 = Z^{-1} * W * WG * WA * E^{-1} \\
M3&: \quad T6 = Z^{-1} * W * WG * E^{-1} \\
M4&: \quad T6 = Z^{-1} * W * WG * WD * E^{-1} \\
M5&: \quad T6 = Z^{-1} * CM * C * CA * WG * E^{-1} \\
M6&: \quad T6 = Z^{-1} * CM * C * CC * WG * E^{-1} \\
M7&: \quad T6 = Z^{-1} * CM * C * CI * WG * E^{-1} \\
M8&: \quad T6 = Z^{-1} * CM * C * CD * WG * E^{-1} \\
M9&: \quad T6 = Z^{-1} * WDUMP * E^{-1}
\end{aligned}$$

Note that $WA$, $WD$, $CA$, $CI$, and $CD$ are all translation transformations concerned with approaching and departing a particular object. In order to allow smooth approach and departure trajectories, these translation distances are iterated from zero to some maximum value or from some maximum value to zero (in integer intervals) depending on whether the effector is approaching or departing. For example: ${}^{WG}WA$ is the approach position of the end-effector before grasping the wire and is (to be) defined as a translation, in the negative $Z$-direction of the $WG$

frame, of the approach distance *z_approach*, say. Thus:

$$^{WG}WA = Trans(0, 0, -(z\_approach))$$

where:

$$\begin{aligned} z\_approach = {} & z\_approach\_initial \\ & z\_approach\_initial - delta \\ & z\_approach\_initial - 2 * delta \\ & \vdots \\ & 0 \end{aligned}$$

*It should be noted well that this type of explicit point-to-point approximation of continuous path control would not normally be necessary with a commercial industrial robot programming language since they usually provide facilities for specifying the end-effector trajectory.*

To complete the task specification, we now have to define the rotations and translations associated with these transforms/frames. Most can be determined by empirical methods, embedding a frame in an object and measuring the object position and orientation. Others, *W* and *WG* in particular, are defined here and their components determined by visual means at run time.

☐ *Z: The position of the position of manipulator with respect to the base coordinate reference frame*

We will assume that the base coordinate system is aligned with the frame embedded in the manipulator base, as shown in Figure 8.5. Thus:

$$Z = I = \text{Identity transform}$$

Note that the frame defining the manipulator base is dependent on the kinematic model of the robot manipulator.

☐ *T6: the position of the end of the manipulator with respect to its base at Z*

The *T6* frame, shown in Figure 8.6, is a computable function of the joint variables. Again, the frame which defines the end of the manipulator is based on the kinematic model. However, there is a convention that the frame should be embedded in the manipulator with the origin at the wrist, with the *Z*-axis directed outward from the wrist to the gripper, with the *Y*-axis directed in the plane of movement of the gripper when it is opening and closing, and with the *X*-axis making up a right-hand system. This is shown more clearly in Figure 8.7 which depicts a more common two-finger parallel jaw gripper.

It is also worth noting that, although we will specify the orientation of *T6* by solving for it in terms of other frames/transforms in the task specification, there is a commonly used convention for specifying the orientation of objects, in general, and *T6* in particular. This convention identifies three rotations about the station

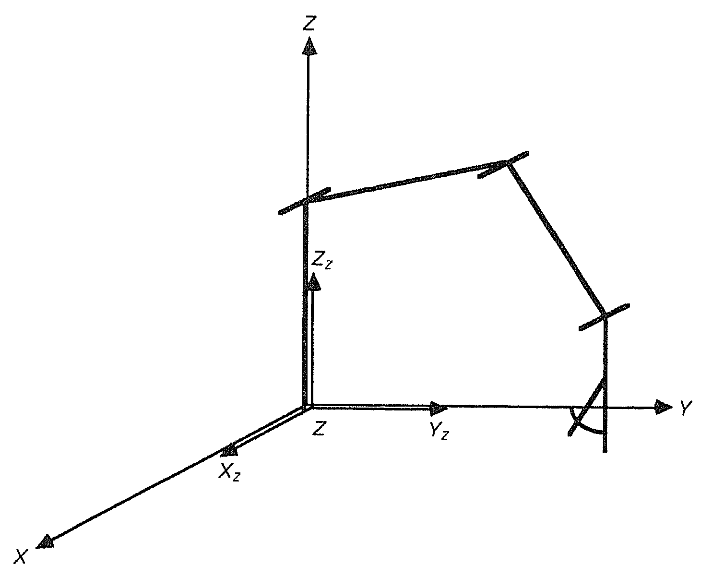


**Figure 8.5** *Z* – the base of the manipulator.

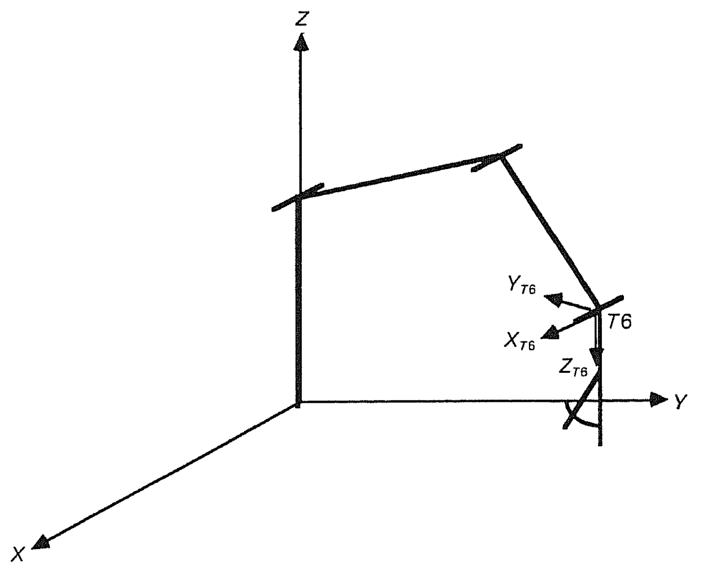


**Figure 8.6** *T6* – the end of the manipulator.

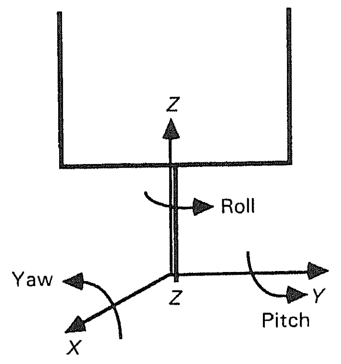


**Figure 8.7** Specifying the orientation of *T6* using roll, pitch, and yaw angles.

coordinate frame embedded in the object which are applied in turn and in a specified order. The rotations are referred to as a *roll* of $\phi$ degrees about the station *Z*-axis, a *pitch* of $\theta$ degrees about the station *Y*-axis, and *yaw* of $\psi$ degrees about the station *X*-axis. The order of rotation is specified as:

$$RPY(\phi, \theta, \psi) = Rot(Z, \phi) Rot(Y, \theta) Rot(X, \psi)$$

Thus, the object is first rotated $\phi^\circ$ about the *Z*-axis, then $\theta^\circ$ about the current station *Y*-axis, and finally $\psi^\circ$ about the station *X*-axis; refer again to Figure 8.7.

☐ *E: the position of the end-effector with respect to the end of the manipulator, i.e. with respect to T6*

The frame *E* representing a special-purpose end-effector for grasping wires is embedded in the tip of the effector, as shown in Figure 8.8, and hence is defined by a translation 209 mm along the *Z*-axis of the *T6* frame and a translation of $-15$ mm along the *Y*-axis of the *T6* frame. Thus:

$$^{T6}E = Trans(0, -15, 209)$$

☐ *OOV: the position of the end-effector out of the field of view of the camera*

This position is defined, with respect to the base coordinate system, such that the end-effector is directed vertically downwards, as shown in Figure 8.9. Thus *OOV* is defined by a translation (of the origin) to the point given by the coordinates (150, 300, 150) followed by a rotation of $-180^\circ$ about the station *X*-axis:

$$OOV = Trans(150, 300, 150)\ Rot(X, -180)$$

☐ *CEN: the position of the end-effector centred over the tray, defined with respect to the base coordinate reference frame*

This position is defined such that the end-effector is directed vertically downwards

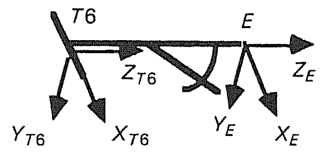


**Figure 8.8** *E* – the end-effector.

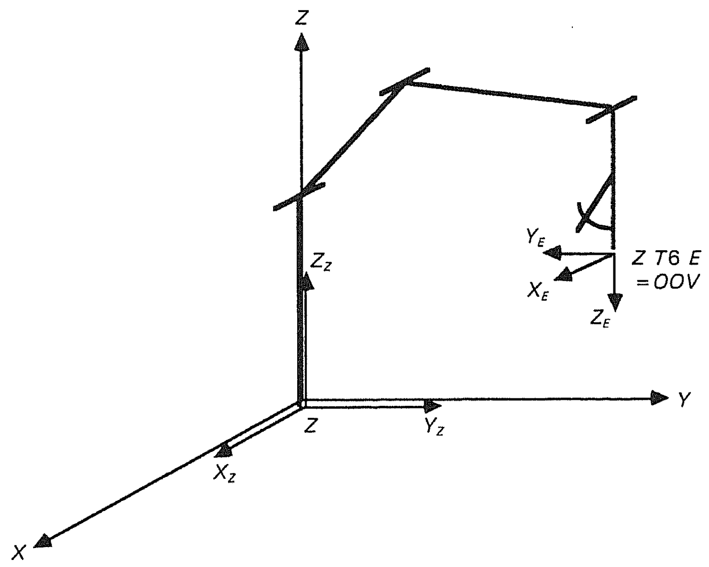


**Figure 8.9** *OOV* – the end-effector out of the camera's field of view.

over the centre of the tray, as shown in Figure 8.10. Thus, *CEN* is defined by a translation (of the frame origin) to the point given by the coordinates (0, 360, 150) followed by a rotation of $-180^\circ$ above the station *X*-axis:

$$CEN = Trans(0, 360, 150)\ Rot(X, -180)$$

☐ *WDUMP: the position of the end-effector over the bin of crimped wires, defined with respect to the base coordinate reference frame*

This position is defined such that the end-effector is directed $-45^\circ$ to the horizontal over the centre of a bin as shown in Figure 8.11. Thus, *WDUMP* is defined by a translation of the frame origin to the point given by the coordinates (0, 500, 160) followed by a rotation of $-135^\circ$ about the station *X*-axis.

$$WDUMP = TRANS(0, 500, 160)\ Rot(X, -135)$$

☐ *W: the position of the wire end, defined with respect to the base coordinate reference frame*

The origin of the wire frame *W* is defined to be at the end of the wire, with its

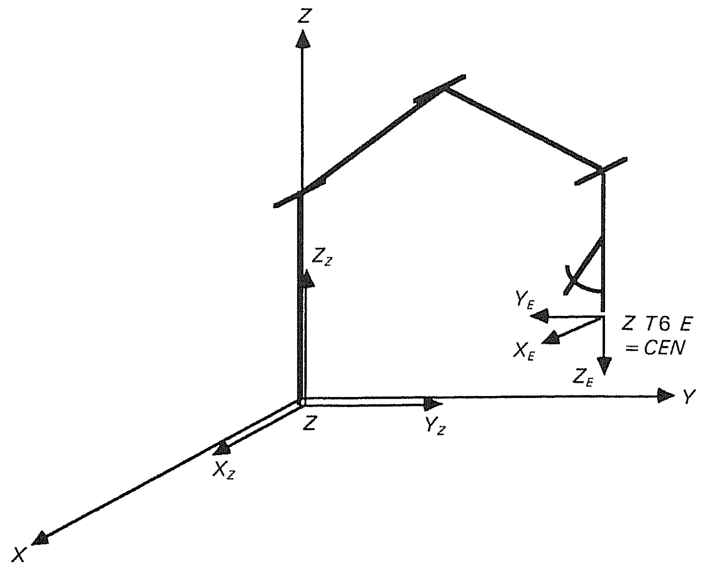


Figure 8.10 *CEN* – the end-effector centred over the tray of wires.

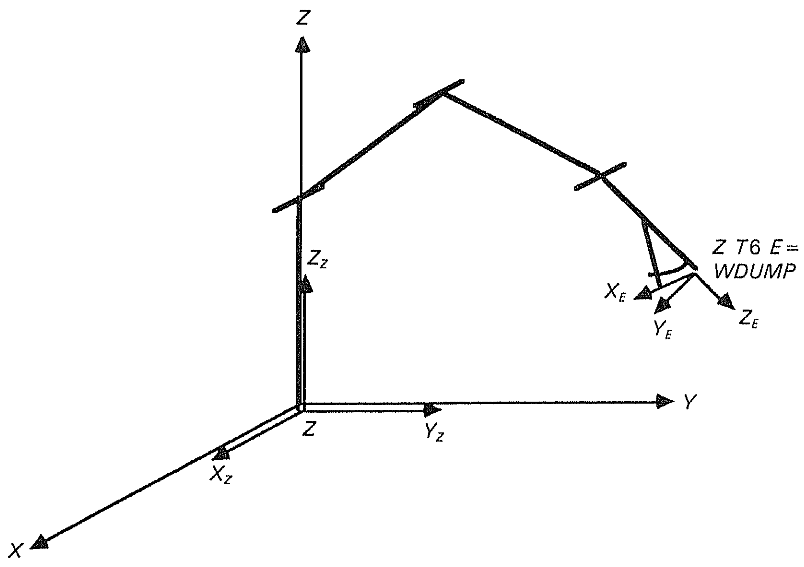


Figure 8.11 *WDUMP* – the end-effector over the bin of crimped wires.

$Z$-axis aligned with the wire's axis of symmetry. The $X$-axis is defined to be normal to the tray on which the wires lie, directed vertically upwards. The $Y$-axis makes up a right-hand system. Since we are assuming that the wires are lying flat on the tray, both the $X$- and $Y$-axes lie in the plane of the tray. Furthermore, we will assume that the tray lies in the $X$–$Y$ plane of the base reference frame. Thus:

$$W = Trans(x, y, 0)Rot(z, \theta)Rot(y, -90)$$

It is the responsibility of the vision system to analyse the image of the wires and to generate the components of this frame automatically, specifically by computing $x, y$, and $\theta$. $W$ is illustrated in Figure 8.12.

□ *WG: the position of the end-effector holding the wire, defined with respect to the wire end*

The origin of the wire gripper frame $WG$ is defined to be located a short distance from the origin of $W$, on the wire's axis of symmetry. The $Z$-axis is defined to be normal to the plane of the tray, directed downwards. The $Y$-axis is defined to be normal to the axis of symmetry of the wire, in the plane of the tray. The $Y$-axis makes up a right-hand system. $WG$ is illustrated in Figure 8.13.

*It is important to note that we define the WG frame in this manner since this*

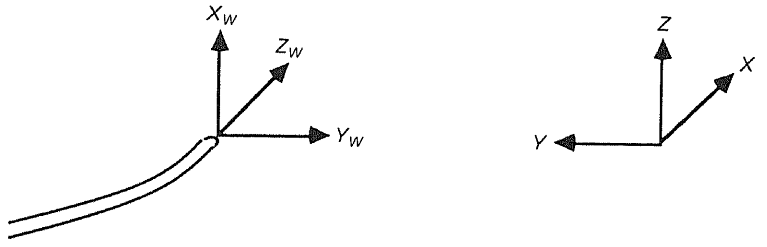


Figure 8.12 *W* – the position of the wire-end.

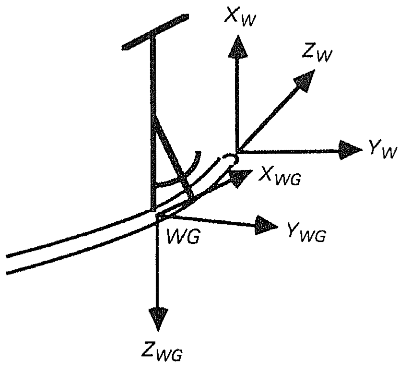


Figure 8.13 *WG* – the wire grasp position.

*is how the end-effector E will be oriented when grasping the wire, i.e. with the Z-axis pointing vertically downwards and the Y-axis at right-angles to the wire.*

As with $W$, the vision system must return a homogeneous transformation defining this frame; we cannot assume that $WG$ will be a fixed offset from $W$ since we are assuming that the curvature of the wire near the end will vary from wire to wire.

□ *WA: the position of the end-effector approaching the grasp position, defined with respect to the wire-grasp position*

This is defined to be a position directly above the wire grasp point. As such, it simply involves a translation in the negative direction of the $Z$-axis of the $WG$ frame. Since it is wished to approach the wire along a known path, many approach positions are used in which the translation distances get successively smaller. This motion, then, approximates continuous path control. Thus:

$$WA = Trans(0, 0, -(z\_approach)),$$

where

$$\begin{aligned} z\_approach = {} & z\_approach\_initial \\ & z\_approach\_initial - delta \\ & z\_approach\_initial - 2 * delta \\ & \quad \vdots \\ & \quad 0 \end{aligned}$$

$WA$ is illustrated in Figure 8.14.

□ *WD: the position of the end-effector departing the grasp position (having grasped the wire), defined with respect to the original wire-grasp position*

In a similar manner to $WA$, $WD$ is defined as a translation in the negative direction of the $Z$-axis of the $WG$ frame, except that in this case the translation distance becomes successively greater. Hence:

$$WD = Trans(0, 0, -(z\_depart)),$$

where:

$$\begin{aligned} z\_depart = {} & 0, \\ & delta, \\ & 2 * delta, \\ & \quad \vdots \\ & z\_depart\_final \end{aligned}$$

□ *CM: the position of the crimping machine, defined with respect to the base coordinate reference system*

The frame $CM$, representing the crimping machine, is defined to be embedded in

a corner of the machine, as shown in Figure 8.15. Thus:

$$CM = Trans(150, 300, 0)$$

Note that the coordinates of the origin of $CM$, (150, 300, 0), are determined empirically.

□ *C: the position of the crimp (ready to be attached), defined with respect to the crimping machine*

The origin of $C$, representing the crimp, is defined to be at the front of the crimp, of the radial axis; the $Z$-axis is defined to be coincident with the radial axis (directed in toward the crimp), the $X$-axis is defined to be directed vertically upward, and the $Y$-axis makes a right-hand system; see Figure 8.16. Thus:

$$C = Trans(40, 40, 65)Rot(Y, 90)Rot(Y, 180)$$

□ *CA: the position of the wire end approaching the crimp, defined with respect to the crimp*

$CA$ is a frame embedded in the end of the wire, in exactly the same manner as $W$, except that it is positioned in front of, i.e. approaching, the crimp. Thus, as shown in Figure 8.17, $CA$ simply involves a translation of some approach distance in the negative direction of the $Z$-axis of $C$.

Since, in a similar manner to $WA$, we want to approach the crimp along a known path, many approach positions are used such that the translation distance gets successively smaller.

Thus:

$$CA = Trans(0, 0, -(z\_approach)),$$

where:

$$\begin{aligned} z\_approach = {} & z\_approach\_initial \\ & z\_approach\_initial - delta, \\ & z\_approach\_initial - 2 * delta \\ & \quad \vdots \\ & \quad 0 \end{aligned}$$

□ *CC: the position of the wire end in contact with the crimp, defined with respect to the crimp*

Since the frames embedded in the end of the wire and the frame embedded in the crimp align when the wire is in contact with the crimp, this transform is simply the identity transform. Had either of these two frames been defined differently, CC would have been used to define the relationship between the end of the wire and the crimp, which would be, in effect, a series of rotations.

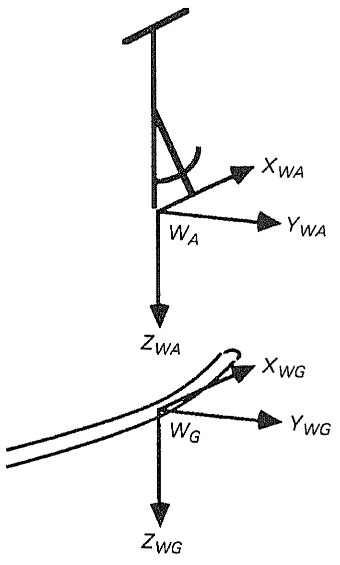


**Figure 8.14** *WA* – the position of the end-effector approaching the grasp position.

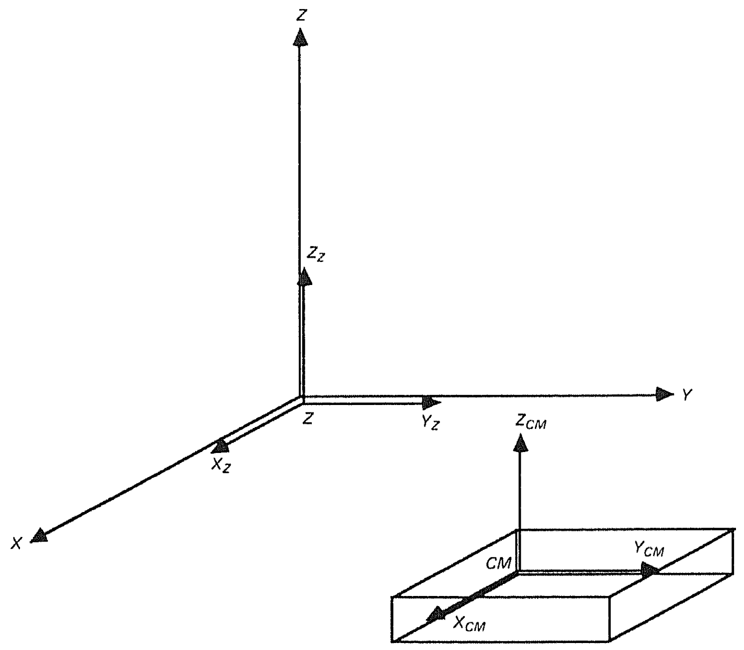


**Figure 8.15** *CM* – the position of the crimping machine.

☐ *CI: the position of the wire end inserted in the crimp, defined with respect to the crimp*

*CI* is a frame embedded in the end of the wire, in exactly the same manner as *CA*, except that it represents the wire inserted in the crimp. Thus, *CI* simply involves a translation of some insertion distance in the positive direction of the *Z*-axis of *C*.

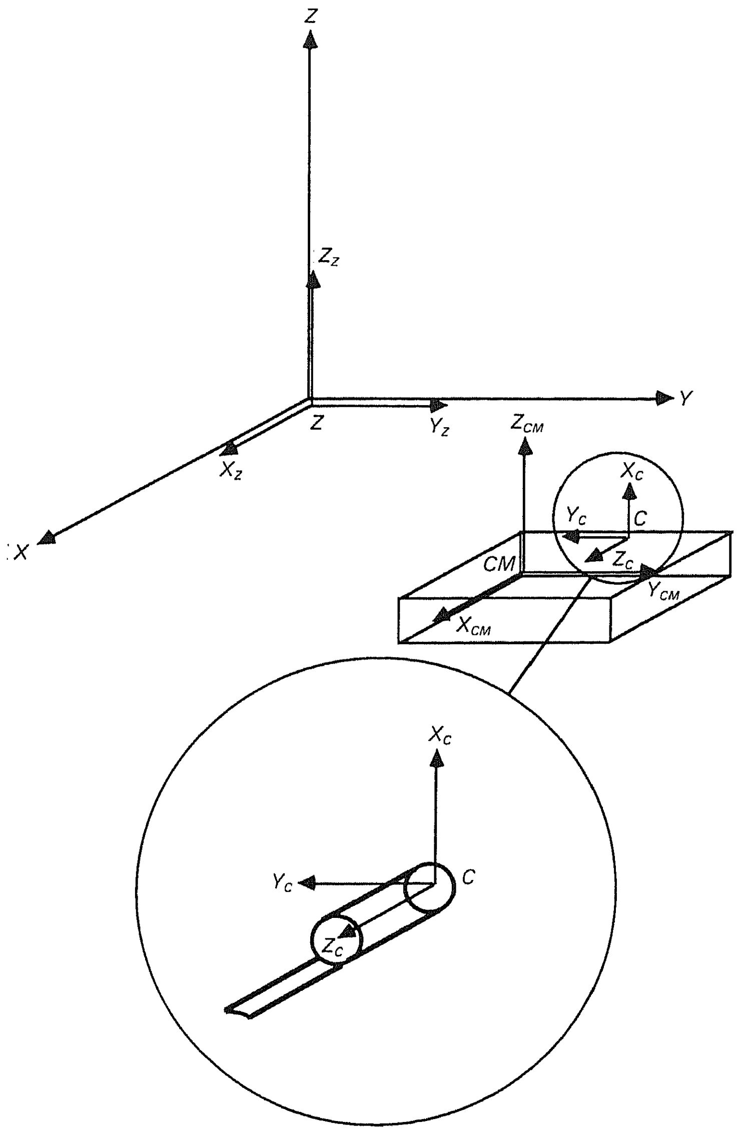


**Figure 8.16** *C* – the position of the crimp.

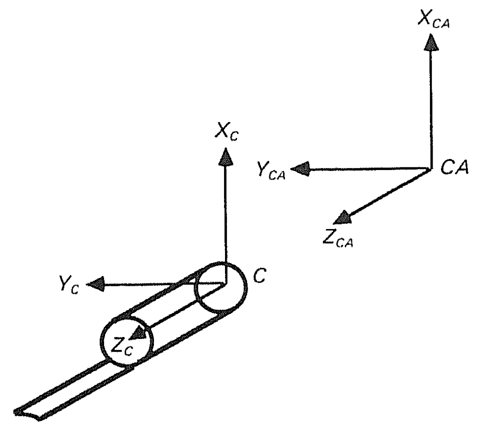


**Figure 8.17** *CA* – the position of the wire end approaching the crimp.

In a similar manner to *CA*, many insertion positions are used such that the translation distances get successively greater. Thus:

$$CI = Trans(0, 0, z\_insert),$$

where:

$$\begin{aligned} z\_insert = {} & 0, \\ & delta, \\ & 2 * delta, \\ & \vdots \\ & z\_insert\_final \end{aligned}$$

☐ *CD: the position of the wire end departed from the crimping machine (the crimp having been attached), defined with respect to the crimp*

In a similar manner to *CA*, *CD* is defined as a translation in the negative direction of the *Z*-axis of the *C* frame, except that in this case the translation distance becomes successively greater. Hence:

$$CD = Trans(0, 0, -(z\_depart)),$$

where:

$$\begin{aligned} z\_depart = {} & 0, \\ & delta, \\ & 2 * delta, \\ & \vdots \\ & z\_depart\_final \end{aligned}$$

The task specification is now complete and it simply remains to program the robot by implementing these transform equations. We will accomplish this in terms of a simple robot programming language described in the next section.

## 8.4 A simple robot-programming language

We introduce here a very simple robot-programming language to illustrate how this manipulator task might be coded. The robot language, RCL, is not intended to be a fully fledged programming language. It is, rather, intended to facilitate the direct implementation of the manipulator task specifications, described in the preceding section, in a structured programming environment. As such, it is intended to facilitate the representation of coordinate frames and computations on frames, to provide an elegant and simple interface to robot vision facilities, and to provide structured programming control constructs. RCL is an interpretative language, implemented using a simple recursive descent algorithm. The philosophy behind the specification of the language syntax is that an RCL program should be almost identical in appearance to the task specification. Thus, both arithmetic and frame/transform expressions are allowed; a built-in frame data-type is provided and several predefined functions are provided for the specification of translations and rotations. The robot vision interface is facilitated by providing two built-in functions which return, among other things, frames defining the position and orientation of the required objects. A complete RCL program comprises two parts: a frame variable definition part and a series of statements. These statements may be either arithmetic expression statements, frame expression statements, built-in functional primitives, or structures programming control construct statements. Since the language components divide naturally into these five distinct sections, each of these topics will be described and discussed in turn.

☐ *Data-types and variable declarations*

There are only two data-types in RCL: an integer type and a predefined frames type. Variables of integer type are declared implicitly when the variable identifier is used in the program; there is no need (indeed, no facility exists) to declare integer variables explicitly in the program. However, variables of the frame type must be explicitly declared at the beginning of the program. Frame variables are functionally important and this is recognized by the requirement to define them explicitly. Frame variables are declared in the frame declaration part, introduced by the keyword FRAME, and by listing all the required frame variables. Since frame variables have such an important and central function in the program, they are distinguished by a leading character ^ i.e. all frame variables begin with the character ^.

☐ *Functional primitives*

Several built-in functional primitives have been incorporated in RCL. These functions are broadly concerned with the three categories of system initialization, robot motion, and visual sensing. The system initialization primitives include the functions `LOAD_ROB` which loads the robot parameters from file allowing RCL to control two different robots, and `LOAD_CAM` which loads the twelve coefficients of the camera model (to be discussed in Section 8.6) from file. The robot motion primitives include the functions GRASP, RELEASE, HOME, DELAY, and MOVE. The GRASP and RELEASE functions simply cause the end-effector gripper to close and open fully; the HOME function causes the robot manipulator to return to a predefined home position; and the DELAY function causes the robot program to be suspended for a specified period of time. The MOVE function accepts the one parameter, a frame expression, and causes the robot manipulator to move to a particular position and orientation as specified by the *T6* frame definition given by the frame expression parameter. Thus, in typical situations, the *T6* frame is assigned the value of transform/frame equation (as discussed in the preceding section on task specification) and the appropriate manipulator movement is effected by passing this *T6* variable to the MOVE function.

The visual sensing primitive of interest here is the function WIRE which provides the interface to the robot vision sub-system, specifically to determine the position and orientation of a suitable wire for grasping, based on the heuristic grey-scale image analysis techniques detailed in Section 8.5. It returns two frame variables corresponding to the frames *W* and *WG* of the wire-crimping task discussed above.

### □ *Arithmetic expression statements*

This type of statement simply provides the facility to evaluate integer expressions, involving any or all of the standard multiplication, division, addition, and division operators (*, /, +, and −, respectively) and to assign the value of this expression to an integer variable. This expression may be parenthesized and the normal precedence relations apply. The expression operands may be either integer variables or integer constant values.

### □ *Frame expression statements*

The frame expression statement type is a central feature of RCL. It allows frame variables and frame functions to be combined by homogeneous transformation matrix multiplication, represented in RCL by the infix operator *, and the resultant value to be assigned to a frame variable. Additionally, frame expressions can be used directly as parameters in the MOVE function. The syntax of the frame statement, expressed in Backus–Naur form, is as follows:

```
<frame_statement>  ::= <frame_variable>:=<frame_expression>
<frame_expression>::= <frame_entity>      {*<frame_entity>}
<frame_entity>     ::= <frame_variable>   | <frame_function>
<frame_function>   ::= <inv_function>     |
                       <rotx_function>    |
                       <roty_function>    |
                       <rotz_function>    |
                       <rpy_function>     |
                       <trans_function>   |
```

Thus, the frame expression allows the combination of a frame variable and/or any of the six in-built frame functions. These functions, INV, ROTX, ROTY, ROTZ, RPY and TRANS, implement transforms corresponding to a homogeneous transformation inversion, rotation about the *x*-, *y*-, and *z*-axes, manipulator orientation specification using the standard roll, pitch and yaw convention, and translation, respectively.

The INV function takes one parameter, a frame expression, and returns frame value equivalent to the inverse of the frame parameter value.

The ROTX, ROTY, and ROTZ functions take one parameter, an integer expression representing the values of the rotation in degrees, and return frame value equivalent to the homogeneous transformation corresponding to this rotation.

The RPY function takes three parameters, all integer expressions, representing the values of the roll, pitch and yaw rotations. Again, it returns frame value equivalent to the homogeneous transformation corresponding to the combination of these rotations.

The TRANS function takes three parameters, again all integer expressions, representing the *x*-, *y*-, and *z*-coordinates to the translation vector. It returns a frame value equivalent to the homogeneous transformation corresponding to this translation.

To illustrate the use of the frame expression, consider the first move in the wire-crimping task. This is represented by a move corresponding to the frame *T6*, given by the expression:

$$T6 = Z^{-1} * OOV * E^{-1}$$

where:

$$Z = \text{identity transform}$$
$$OOV = Trans(150, 300, 150)Rot(x, -180)$$
$$E = Trans(0, -15, 209)$$

Assuming the frames ^T6, ^Z, ^OOV, and ^E have been declared, this is written in RCL as:

```
^Z   := TRANS(0,0,0);
^OOV := TRANS(150,300,150)*ROTX(-180);
^E   := TRANS(0,-15,209);
^T6  := INV(^Z)*OOV*INV(^E);
```

and a move to this position is effected by the statement:

```
MOVE(^T6);
```

### □ *Structured programming control constructs*

The structured programming control constructs include a REPEAT-UNTIL statement, an IF-THEN-ELSE-ENDIF statement, a FOR statement, and a

WHILE statement. These statements adhere to the normal functionality of such structured control constructs.

Since the syntax and semantics of the RCL language are based on the task specification methodology described in detail in the preceding section, the implementation of the wire-crimping task becomes a very simple translation process. This was illustrated in the preceding section by the implementation of the first move of the task. The remaining moves are equally amenable to translation and understanding and the final wire-crimping program follows:

☐ *An RCL implementation of the wire-crimping task*

```
/**************************************************/
/*                                                */
/* RCL Wire Crimping Program For 6R/600 Manipulator */
/*                                                */
/**************************************************/

/* Frame Declarations */

FRAME ^wire,
      ^wiregrasp,
      ^wireapproach,
      ^wiredepart,
      ^crimp,
      ^crimpapproach,
      ^crimpdepart,
      ^crimpcontact,
      ^crimpinsert,
      ^crimpmachine,
      ^wiredump,
      ^centre,
      ^outofview,
      ^Z,
      ^T6,
      ^effector;

/* the position of the manipulator is coincident */

/* with the base coordinate reference frame */

^Z := TRANS(0,0,0);

/* the end-effector is at the tip of the */
/* wire gripper; a position defined with */
/* respect to the end of the manipulator */

ey := -15;
ez := 195;
^effector := TRANS(0,ey,ez);
```

```
/* the position of the end-effector */
/* out of the field-of-view of the camera */

oovx := 150;
oovy := 300;
oovz := 150;
^outofview := TRANS(oovx,oovy,oovz) * RPY(0,0,-180);

/* the position of the end-effector over */
/* the bin of crimped wires              */

wdumpx := 0;
wdumpy := 550;
wdumpz := 160;
^wiredump := TRANS(wdumpx,wdumpy,wdumpz) *
             RPY(0,0,-135);

/* the position of the end-effector */
/* centred over the tray of wires */

centrex := 0;
centrey := 360;
centrez := 150;
^centre := TRANS(centrex,centrey,centrez) *
           RPY(0,0,-180);

/* the position of the crimping machine */

cmx := 150;
cmy := 300;
cmz := 0;
^crimpmachine := TRANS(cmx,cmy,cmz);

/* the position of the crimp, ready to be */
/* attached; as position defined with */
/* respect to the crimping machine */

cx := 55;
cy := 55;
cz := 85;
^crimp := TRANS(cx,cy,cz) * ROTY(90) * ROTZ(180);

/* the position of the wire-end in contact with */
/* the crimp */

^crimpcontact := TRANS(0,0,0);

/* the position of the wire-end inserted in the crimp */
```

```
insertionlength := 5;
^crimpinsert := TRANS(0,0,insertionlength);

/* load the robot model of the Smart Arms */
/* 6R/600 manipulator from file */

LOAD_ROB;

/* load the components of the camera model from file */

LOAD_CAM;

/* incremental distances for point-to-point */
/* approximation to continuous path movement */

delta := 3;

/* time delay between gross manipulator */
/* point-to-point movements */

lag := 20;

REPEAT

 /* move out of the field-of-view of the camera */

 ^T6 := INV(^Z) * ^outofview * INV(^effector);
 MOVE(^T6);
 DELAY(lag);

 /* determine the position and orientation of the */
 /* wire-end and wire grasping point using vision */

 WIRE(^wire,^wiregrasp);

 /* if not error in the robot vision routine */
 /* proceed with the task */

 IF errcode = 0
  THEN

   /* move to a position above the centre of the tray*/

   ^T6 := INV(^Z) * ^centre * INV(^effector);
   MOVE(^T6);
   DELAY(lag);
   RELEASE;
```

```
/* when grasping, the end-effector is defined */
/* to be between the jaws of the gripper */

ey := 5;
^effector := TRANS(0,ey,ez);

/* move to an approach point above the grasp point */

approachdistance := 30;
^wireapproach := TRANS(0,0,-approachdistance);
^T6 := INV(^Z) * ^wire * ^wiregrasp * ^wireapproach
 * INV (^effector);
MOVE(^T6);
DELAY(lag);

/* move to the grasp point through */
/* successive approach points */

approachdistance := approachdistance - delta;

REPEAT
 ^wireapproach := TRANS(0,0,-approachdistance);
 ^T6 := INV(^Z) * ^wire * ^wiregrasp*
  ^wireapproach * INV(^effector);
 MOVE(^T6);
 approachdistance := approachdistance - delta;

UNTIL approachdistance <= 0;

/* move to the final grasp point and grasp the wire */

^T6 := INV(^Z) * ^wire * ^wiregrasp * INV(^effector);
MOVE(^T6);
GRASP;

/* move to the depart position through */
/* successive depart points */

departdistance := delta;

REPEAT
 ^wiredepart := TRANS(0,0,-departdistance);
 ^T6 := INV(^Z) * ^wire * ^wiregrasp * ^wiredepart
  * INV(^effector);
 MOVE(^T6);
 departdistance := departdistance + delta;
UNTIL departdistance > 30;
```

```
approachdistance := 40;

/* the end-effector is defined to be at the */
/* inside of the upper jaw of the gripper */
/* now that the wire has been grasped */

ey := -15;
^effector := TRANS(0,ey,ez);

^crimpapproach := TRANS(0,0,-approachdistance);
/* move to an approach position */
/* in front of the crimp */

^T6 := INV(^Z) * ^crimpmachine * ^crimp *
 ^crimpapproach * ^wiregrasp * INV(^effector);
MOVE(^T6);
DELAY(lag);

/* bring the wire into contact with the crimp */
/* by moving through successive approach points */

approachdistance := approachdistance - delta;

REPEAT
 ^crimpapproach := TRANS(0,0,-approachdistance);
 ^T6 := INV(^Z) * ^crimpmachine * ^crimp *
  ^crimpapproach * ^wiregrasp * INV(^effector);
 MOVE(^T6);
approachdistance := approachdistance - delta;
UNTIL approachdistance <= 0;

/* final contact position */

^T6 := INV(^Z) * ^crimpmachine * ^crimp *
 ^crimpcontact * ^wiregrasp * INV(^effector);
MOVE (^T6);

/* insert wire in crimp */

^T6 := INV(^Z) * ^crimpmachine * ^crimp *
 ^crimpinsert * ^wiregrasp * INV(^effector);
MOVE(^T6);

/* actuate the crimping machine */
/* * this is a virtual action * */

DELAY(lag);
```

```
  /* withdraw with the crimped wire */
  /* through successive depart positions */

  departdistance := delta;

  REPEAT
   ^crimpdepart := TRANS(0,0,-departdistance);
   ^T6 := INV(^Z) * ^crimpmachine * ^crimp *
    ^crimpdepart * ^wiregrasp * INV(^effector);
   MOVE(^T6);
   departdistance := departdistance + delta;
  UNTIL departdistance > 35;

  /* move to a position above the collection bin */

  ^T6 := INV(^Z) * ^wiredump * INV(^effector);
  MOVE(^T6);
  DELAY(lag);

  RELEASE;

  /* return to the position above the */
  /* centre of the tray */

  ^T6 := INV(^Z) * ^centre * INV(^effector);
  MOVE(^T6);
  DELAY)lag);

 ENDIF;

/* this is repeated until there are no more wires */
/* to be crimped; WIRE returns error code 20 */

UNTIL errcode = 20;
```

## 8.5 Two vision algorithms for identifying ends of wires

### 8.5.1 A binary vision algorithm

The main problem in this application is to identify the position and orientation of both a wire end and of a suitable grasp point to allow the robot manipulator to pick up the wire and insert it in a crimp. If we assume that the wires are well-scattered and lie no more than one or two deep, then all the requisite information may be gleaned from the silhouette of the wire and, hence, binary vision techniques can be used. In order to facilitate simple image analysis, we threshold the image and thin the resultant segmented binary image.

The original image is acquired at the conventional 512 × 512 pixel resolution with eight-bit grey-scale resolution (see Figure 8.18). To ensure fast processing and analysis, we first reduce the resolution to 128 × 128 pixels (see Figure 8.19) resulting in a reduction of the complexity of subsequent operations by a factor of sixteen. This reduction is important as the computational complexity of thinning operations is significant. There are essentially two ways in which this reduced resolution image may be generated: by sub-sampling the original image every fourth column and every fourth line or by evaluating the average of pixel values in a 4 × 4 window. As

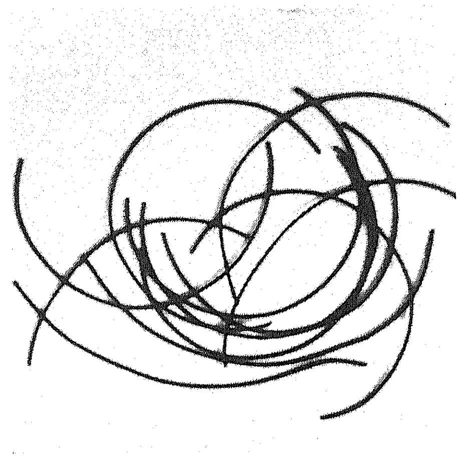


**Figure 8.18** 512 × 512 image.

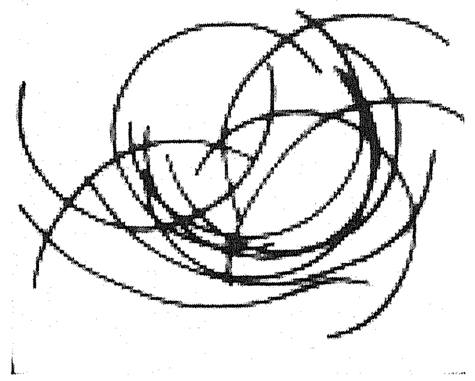


**Figure 8.19** 128 × 128 image.

we saw in Chapter 4, it is desirable to reduce the image noise and, since this can be accomplished by local averaging, the reduced resolution image in this implementation is generated by evaluating the local average of a 4 × 4 (non-overlapping) region in the 512 × 512 image. This also minimizes the degradation in image quality (referred to as *aliasing*) which would result from sub-sampling.

The image is then segmented by thresholding in the manner discussed in Chapter 4; the threshold value is automatically selected using the approach associated with the Marr–Hildreth edge detector (see Section 4.1.9). Figure 8.20 shows the binary image generated by thresholding the original grey-scale image at the automatically determined threshold.

Once the binary image has been generated, the next step is to model the wires in some simple manner. The skeleton is a suitable representation of electrical wires, objects which display obvious axial symmetry. In this instance, we use the thinning technique described in Section 4.2.3; Figure 8.21 illustrates the application of this thinning algorithm to the binary image shown in Figure 8.20.

Having processed the image, we now proceed to its analysis. There are essentially two features that need to be extracted from the image:

(a) the position of a point at which the robot end-effector should grasp the wire and the orientation of this point on that wire;
(b) the position and orientation of the wire end in relation to the point at which the wire is to be grasped.

The orientations are required because unless the wire is gripped at right-angles to the tangent at the grasp point, the wire will rotate in compliance with the finger grasping force. The orientation of the endpoint is important when inserting the wire in the crimping-press as the wire is introduced along a path coincident with the tangent to the wire at the endpoint. Based on the skeleton model of the wires, a wire

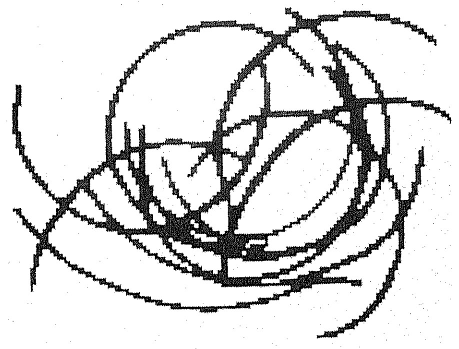


**Figure 8.20** Binary image.

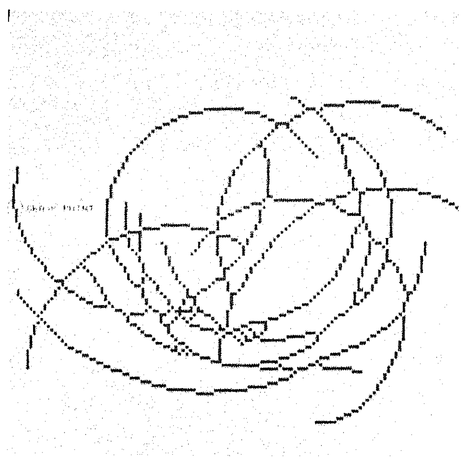


Figure 8.21 Thinned image.

segment may be defined as a subsection of a wire bounded at each end by either a wire-crossing or by an arc-end (wire segment end). Thus, a wire segment with two valid endpoints, at least one of which is an arc-end, and with a length greater than some predefined system tolerance, contains a feasible grasp point. This is a point some suitable fixed distance (15 mm) from the wire end.

Once the positions of both the grasp-point and the endpoint are known, the orientations or tangential angles of these two points are estimated. The tangent to the wire at the grasp-point is assumed to be parallel to the line joining two skeletal points equally displaced by two pixels on either side of the grasp-point. The tangent to the wire end is assumed to be parallel to a line joining the endpoint and a skeletal point three pixels from the end. Both of these tangential angles are estimated using the world coordinates corresponding to these pixel positions; these world coordinates are obtained using the camera model and inverse perspective transformation to be described in Section 8.6.

A typical selected grasp-point is shown in the thinned image (Figure 8.21) of the original image of a tray of wires (Figure 8.20).

### *8.5.2 A grey-scale vision algorithm*

If the organization of the wires becomes more complex than assumed in the preceding section, with many layers of wires occluding both themselves and the background, the required information can no longer be extracted with binary imaging techniques. The grey-scale vision system described in this section addresses these issues and facilitates analysis of poor contrast images. It is organized as two levels, comprising a peripheral level and a supervisory level. All shape identification and analysis is based on boundary descriptors built dynamically by the peripheral

level. The supervisory level is responsible for overall scheduling of activity, shape description, and shape matching. The use of an area-of-interest operator facilitates efficient image analysis by confining attention to specific high-interest sub-areas in the image. Thus, the algorithm described here uses three key ideas: dynamic boundary following (see Chapter 5); planning based on reduced resolution images, and a two-level organization based on peripheral and supervisory hierarchical architecture. These three ideas facilitate efficient analysis and compensate for the additional computational complexity of grey-scale techniques. The system is based on $256 \times 256$ pixel resolution images; the reduced resolution image is generated by local averaging in every $2 \times 2$ non-overlapping region in the acquired $512 \times 512$ image. The choice of resolution was based on a consideration of the smallest objects that need to be resolved and the minimum resolution required to represent these objects.

#### □ *The peripheral level*

The peripheral level corresponds to conventional low-level visual processing, specifically edge detection and the generation of edge and grey-scale information at several resolutions, and segmentation using boundary detection. The Prewitt gradient-based edge operator described in Chapter 5 is used as it provides reasonable edges with minimal computational overhead, especially in comparison to other edge operators. The edge detector operates on both $256 \times 256$ and $64 \times 64$ resolution images. High-resolution edge detection is used for image segmentation and low-resolution edge detection is used by an area-of-interest operator.

The ability of any edge detector to segment an image depends on the size of the objects in the image with respect to the spatial resolution of the imaging system. The system must be capable of explicitly representing the features (edges) that define the objects, in this case electrical wires. When dealing with long cylinder-like objects, the constraining object dimension is the cylinder diameter. At least three pixels are required to represent the wire (across the diameter) unambiguously: one for each edge and for the wire body. Using wires of diameter 1.0 mm imposes a minimum spatial resolution of 2 pixels/mm or a resolution of $256 \times 256$ for a field of view of $128 \times 128$ mm. Using a spatial resolution of 1 pixel/mm will tend to smear the object (given that we are reducing the resolution by local averaging and not by sub-sampling). Edge detection tests at this resolution showed that such smearing does not adversely affect the boundary/feature extraction performance if the wire is isolated (i.e. the background is clearly visible) but in regions of high occlusion where there are many wires in close proximity to the edge or boundary, quality does degrade significantly. Tests using a spatial resolution of 0.5 pixels/mm indicated that a detector's ability to segment the image reliably is severely impaired in most situations.

There are several approaches which may be taken to boundary building; this system uses a dynamic contour following algorithm and is the same one described in detail in Chapter 5. As the algorithm traces around the boundary, it builds a boundary chain code (BCC) representation of the contour; see Chapter 7. The complete BCC represents the segmented object boundary and is then passed to the supervisory level for analysis. Figure 8.22 illustrates the boundary following process at various points along the wire contours. The disadvantage of the contour following technique is that, because the algorithm operates exclusively on a local basis using no *a priori* information, the resulting segmentation may not always be reliable and the resulting contour may not correspond to the actual object boundary. In particular, the presence of shading and shadows tends to confuse the algorithm.

The boundary following algorithm, which effects the object's segmentation, is guided by processes at the supervisory level on two distinct bases. Firstly, the supervisory level defines a sub-section of the entire image to which the boundary following process is restricted: this sub-area is effectively a region within the image in which the vision system has high interest. Secondly, the supervisory level supplies the coordinates of a point at which the boundary following procedure should begin. This is typically on the boundary of the object to be segmented.

☐ *The supervisory level*

The supervisory phase is concerned with overall scheduling of activity within the vision system and with the transformation and analysis of the boundaries passed to it by the peripheral level.

In guiding the peripheral level, its operation is confined to specific areas of high interest and it is supplied, by the supervisory level, with start coordinates for the boundary following algorithm. An interest operator was used which identifies

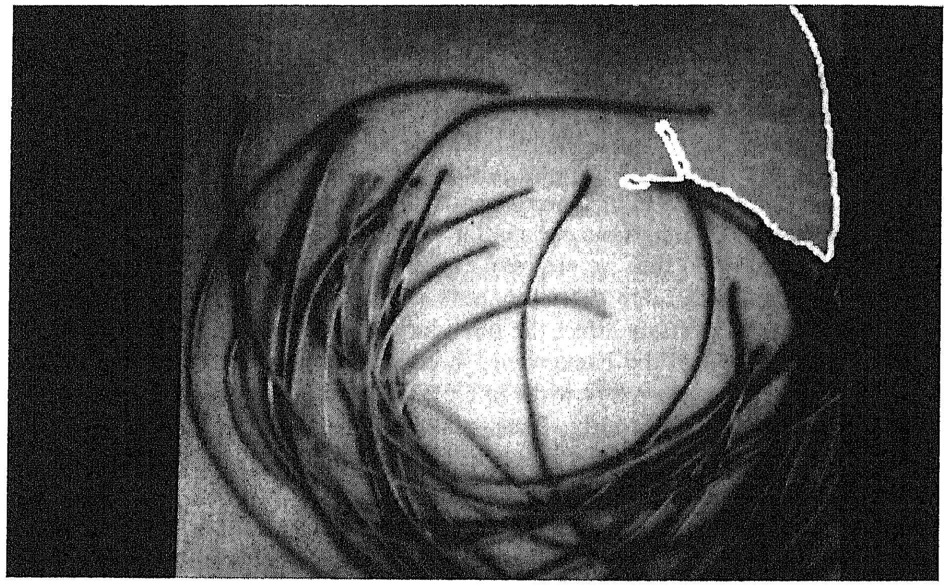


**Figure 8.22** Boundary following.

a sequence of sub-areas within the image, ordered in descending levels of interest. This operator is based on the analysis of the edge activity in a reduced resolution image and allows the system to avoid cluttered areas with many (occluding) wires and concentrate on points of low scene population which are more likely to contain isolated and accessible wires. The area of interest is one-sixteenth of the size of the original image and is based on a $4 \times 4$ division of a $64 \times 64$ pixel resolution image.

The approach taken to the wire-crimping application is to extract a contour, representing the boundary of a group of wires, in a specific area of interest in the image and to analyse this boundary to determine whether or not it contains a boundary segment describing a wire end. What is required of the supervisory processes is to ascertain which part of the contour, if any, corresponds to the wire-end template and subsequently to determine the position and orientation of both the end of the wire and a suitable grasp-point. As we noted in Chapter 7, the use of BCC-based shape descriptors to identify shapes is not reliable and, instead, the wire end is identified by heuristic analysis, formulated as follows.

A boundary segment characterizing a wire end is defined to be a short segment (20 units in length) in which the boundary direction at one end differs by 180° from the direction at the other end, and in which the distance between the endpoints is less than or equal to 5 units. In addition, the wire end should be isolated, i.e. there should be no neighbouring wires which might foul the robot end-effector when grasping the wire. This condition is identified by checking that the edge magnitude in the low-resolution image in a direction normal to the boundary direction is less than the usual threshold used by the edge detection process. Figure 8.23 illustrates a wire end extracted from a boundary using this heuristic technique.

### 8.5.3 The vision/manipulator interface

Once the wire end shape has been identified, it is necessary to determine the components of the homogeneous transformations representing the two frames, $W$ and $WG$, which denote the position and orientation of the wire end and the position and orientation of the grasp position with respect to the wire end. In the task specification discussed above, we defined the origin of the wire frame $W$ to be at the end of the wire, with its $Z$-axis aligned with the wire's axis of symmetry, directed away from the end. The $X$-axis of $W$ was defined to be normal to the tray on which the wires lie (and, hence, is normal to image plane) directed vertically upwards. The $Y$-axis makes up a right-hand system. The origin of the wire gripper frame $WG$ was defined to be located on the $Z$-axis of the $W$ frame, in the negative $Z$-direction, and located a short distance from the origin of $W$. The $Z$-axis of $WG$ is defined to be normal to the plane of the tray, directed downwards. The $Y$-axis is defined to be normal to the axis symmetry of the wire, in the plane of the tray. The $X$-axis makes up a right-hand system. Refer again to Figures 8.11 and 8.12.

We can see that, to determine the components of the frame $W$, we only need to identify the position of the end of the wire and orientation of the axis of symmetry of the wire at its end, which gives us the direction of the $Z$-axis. The

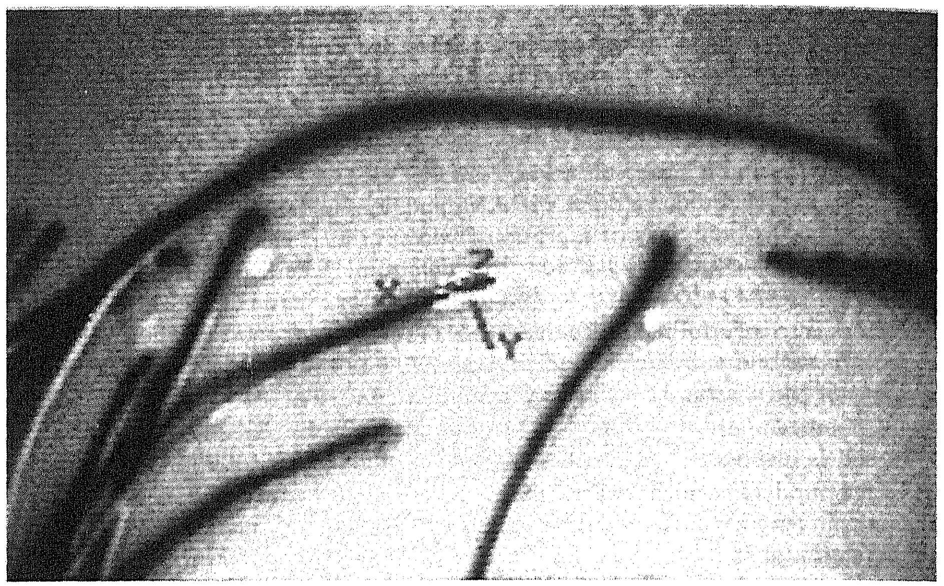


**Figure 8.23** Identification of a wire-end (with *W* frame attached).

*Y*-axis is at right-angles to it and the *X*-axis has already been defined. Similarly, we only need to identify the orientation of the axis of symmetry of the wire at the grasp-point to determine *WG*; this gives us the direction of the *X*-axis; the *Y*-axis is at right-angles to it and the *Z*-axis has already been defined.

The main problem at this stage is that any orientation and position will be computed in the image frame of reference, i.e. using pixel coordinates. This is not satisfactory since the robot task specification is formulated in the real-world frame of reference. Obviously, the relationship between these two reference frames must be established. Once it is, we can transform the relevant image positions (the end of the wire and other points on its axis) to the real-world frame of reference and then compute the required orientations. This relationship is the subject to which we now turn our attention.

## 8.6 The camera model and the inverse perspective transformation

Generally speaking, when we use machine vision to identify the position and orientation of objects to be manipulated by a robot, we must do so with reference to real-world coordinates, i.e. in the real-world frame of reference. However, all the techniques we have dealt with in preceding chapters have been confined to the image frame of reference; we now need to establish the relationship between this image coordinate reference frame and the real-world coordinate reference frame.

For any given optical configuration, there are two aspects to the relationship: the *camera model*, which maps a three-dimensional world point to its corresponding two-dimensional image point, and the *inverse perspective transformation*, which is used to identify the three-dimensional world point(s) corresponding to a particular two-dimensional image point. Since the imaging process is a projection (from a three-dimensional world to a two-dimensional image), the inverse process, i.e. the inverse perspective transformation, cannot uniquely determine a single world point for a given image point; the inverse perspective tranformation thus maps a two-dimensional image point into a line (an infinite set of points) in the three-dimensional world. However, it does so in a useful and well-constrained manner.

For the following, we will assume that the camera model (and, hence, the inverse perspective transformation) is linear; this treatment closely follows that of Ballard and Brown (1982). Details of non-linear models can be found in the references to camera models in the bibliography at the end of the chapter.

### 8.6.1 The camera model

Let the image points in question be given by the coordinates

$$\begin{bmatrix} U \\ V \end{bmatrix}$$

which, in homogeneous coordinates, is written

$$\begin{bmatrix} u \\ v \\ t \end{bmatrix}.$$

Thus:

$$U = \frac{u}{t}$$

and:

$$V = \frac{v}{\mathrm{t}}$$

Let the desired camera model, a transformation which maps the three-dimensional world point to the corresponding two-dimensional image point, be *C*. Thus:

$$\mathbf{C}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ t \end{bmatrix}$$

Hence $C$ must be a $3 \times 4$ (homogeneous) transformation:

$$C = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix}$$

and:

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ t \end{bmatrix}$$

Expanding this matrix equation, we get:

$$C_{11}x + C_{12}y + C_{13}z + C_{14} = u \tag{1}$$

$$C_{21}x + C_{22}y + C_{23}z + C_{24} = v \tag{2}$$

$$C_{31}x + C_{32}y + C_{33}z + C_{34} = t \tag{3}$$

but:

$$u = Ut$$
$$v = Vt$$

so:

$$u - Ut = 0 \tag{4}$$

$$v - Vt = 0 \tag{5}$$

Substituting (1) and (3) for $u$ and $t$, respectively, in (4) and substituting (2) and (3) for $v$ and $t$, respectively, in (5):

$$C_{11}x + C_{12}y + C_{13}z + C_{14} - UC_{31}x - UC_{32}y - UC_{33}z - UC_{34} = 0 \tag{6}$$

$$C_{21}x + C_{22}y + C_{23}z + C_{24} - VC_{31}x - VC_{32}y - VC_{33}z - VC_{34} = 0 \tag{7}$$

Remember that these two equations arose from the association of a particular world point

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

with a particular and corresponding image point

$$\begin{bmatrix} u \\ v \\ t \end{bmatrix}.$$

If we establish this association (i.e. if we measure the values of $x, y, z, U$, and $V$),

we will have two equations in which the only unknowns are the twelve camera model coefficients (which we require). Since a single observation gives rise to two equations, six observations will produce twelve simultaneous equations which we can solve for the required camera coefficients $C_{ij}$. Before we proceed, however, we need to note that the overall scaling of $C$ is irrelevant due to the homogeneous formulation and, thus, the value of $C_{34}$ may be set arbitrarily to 1 and we can rewrite (6) and (7), completing the equations so that terms for each coefficient of $C$ are included, as follows:

$$C_{11}x + C_{12}y + C_{13}z + C_{14} + C_{21}0 + C_{22}0 + C_{23}0 + C_{24}0 - UC_{31}x - UC_{32}y - UC_{33}z = U$$

$$C_{11}0 + C_{12}0 + C_{13}0 + C_{14}0 + C_{21}x + C_{22}y + C_{23}z + C_{24} - VC_{31}x - VC_{32}y - VC_{33}z = V$$

This reduces the number of unknowns to eleven. For six observations, we now have twelve equations and eleven unknowns: i.e. the system of equations is over-determined. Reformulating the twelve equations in matrix form, we can obtain a least-square-error solution to the system using the pseudo-inverse method which we described in Chapter 4.

Let

$$X = \begin{bmatrix}
x^1 & y^1 & z^1 & 1 & 0 & 0 & 0 & 0 & -U^1x^1 & -U^1y^1 & -U^1z^1 \\
0 & 0 & 0 & 0 & x^1 & y^1 & z^1 & 1 & -V^1x^1 & -V^1y^1 & -V^1z^1 \\
x^2 & y^2 & z^2 & 1 & 0 & 0 & 0 & 0 & -U^2x^2 & -U^2y^2 & -U^2z^2 \\
0 & 0 & 0 & 0 & x^2 & y^2 & z^2 & 1 & -V^2x^2 & -V^2y^2 & -V^2z^2 \\
x^3 & y^3 & z^3 & 1 & 0 & 0 & 0 & 0 & -U^3x^3 & -U^3y^3 & -U^3z^3 \\
0 & 0 & 0 & 0 & x^3 & y^3 & z^3 & 1 & -V^3x^3 & -V^3y^3 & -V^3z^3 \\
x^4 & y^4 & z^4 & 1 & 0 & 0 & 0 & 0 & -U^4x^4 & -U^4y^4 & -U^4z^4 \\
0 & 0 & 0 & 0 & x^4 & y^4 & z^4 & 1 & -V^4x^4 & -V^4y^4 & -V^4z^4 \\
x^5 & y^5 & z^5 & 1 & 0 & 0 & 0 & 0 & -U^5x^5 & -U^5y^5 & -U^5z^5 \\
0 & 0 & 0 & 0 & x^5 & y^5 & z^5 & 1 & -V^5x^5 & -V^5y^5 & -V^5z^5 \\
x^6 & y^6 & z^6 & 1 & 0 & 0 & 0 & 0 & -U^6x^6 & -U^6y^6 & -U^6z^6 \\
0 & 0 & 0 & 0 & x^6 & y^6 & z^6 & 1 & -V^6x^6 & -V^6y^6 & -V^6z^6
\end{bmatrix}$$

$$c = \begin{bmatrix} C_{11} \\ C_{12} \\ C_{13} \\ C_{14} \\ C_{21} \\ C_{22} \\ C_{23} \\ C_{24} \\ C_{31} \\ C_{32} \\ C_{33} \end{bmatrix}$$

$$y = \begin{bmatrix} U^1 \\ V^1 \\ U^2 \\ V^2 \\ U^3 \\ V^3 \\ U^4 \\ V^4 \\ U^5 \\ V^5 \\ U^6 \\ V^6 \end{bmatrix}$$

where the trailing superscript denotes the observation number.

Then:

$$\begin{aligned} c &= (X^T X)^{-1} X^T y \\ &= X^\dagger y \end{aligned}$$

We assumed above that we make six observations to establish the relationship between six sets of image coordinates and six sets of real-world coordinates.* This is, in fact, the central issue in the derivation of the camera model, that is, the identification of a set of corresponding *control points*. There are several approaches. For example, we could present the imaging system with a calibration grid, empirically measure the positions of the grid intersections, and identify the corresponding points in the image, either inactively or automatically. The empirical measurement of these real-world coordinates will be prone to error and this error will be manifested in the resultant camera model. It is better practice to get the robot itself to calibrate the system by fitting it with an end-effector with an accurately located calibration mark (e.g. a cross-hairs or a surveyor's mark) and by programming it to place the mark at a variety of positions in the field of view of the camera system. The main benefit of this approach is that the two components of the manipulation environment, the robot and the vision system, both of which are reasoning about coordinates in the three-dimensional world, are effectively coupled and, if the vision system 'sees' something at a particular location, that is where the robot manipulator will go.

### 8.6.2 The inverse perspective transformation

Once the camera model $C$ has been determined, we are now in a position to determine an expression for the coordinates of a point in the real world in terms of the coordinates of its imaged position.

* In general, it is better to overdetermine the system of equations significantly by generating a larger set of observations than the minimal six.

Recalling equations (1)–(5):

$$\begin{aligned} C_{11}x + C_{12}y + C_{13}z + C_{14} &= u = Ut \\ C_{21}x + C_{22}y + C_{23}z + C_{24} &= v = Vt \\ C_{31}x + C_{32}y + C_{33}z + C_{34} &= t \end{aligned}$$

Substituting the expression for $t$ into the first two equations gives:

$$\begin{aligned} U(C_{31}x + C_{32}y + C_{33}z + C_{34}C_{11}x) &= C_{11}x + C_{12}y + C_{13}z + C_{14} \\ V(C_{31}x + C_{32}y + C_{33}z + C_{34}C_{11}x) &= C_{21}x + C_{22}y + C_{23}z + C_{24} \end{aligned}$$

Hence:

$$\begin{aligned} (C_{11} - UC_{31})x + (C_{12} - UC_{32})y + (C_{13} - UC_{33})z + (C_{14} - UC_{34}) &= 0 \\ (C_{21} - VC_{31})x + (C_{22} - VC_{32})y + (C_{23} - VC_{33})z + (C_{24} - VC_{34}) &= 0 \end{aligned}$$

Letting:

$$\begin{aligned} a_1 &= C_{11} - UC_{31} \\ b_1 &= C_{12} - UC_{32} \\ c_1 &= C_{13} - UC_{33} \\ d_1 &= C_{14} - UC_{34} \end{aligned}$$

and:

$$\begin{aligned} a_2 &= C_{21} - VC_{31} \\ b_2 &= C_{22} - VC_{32} \\ c_2 &= C_{23} - VC_{33} \\ d_2 &= C_{24} - VC_{34} \end{aligned}$$

we have:

$$\begin{aligned} a_1x + b_1y + c_1z + d_1 &= 0 \\ a_2x + b_2y + c_2z + d_2 &= 0 \end{aligned}$$

These two equations are, in effect, equations of two planes; the intersection of these planes determines a line comprising the set of real-world points which project onto the image point

$$\begin{bmatrix} U \\ V \end{bmatrix}.$$

Solving these plane equations simultaneously (in terms of $z$):

$$x = \frac{z(b_1c_2 - b_2c_1) + (b_1d_2 - b_2d_1)}{(a_1b_2 - a_2b_1)}$$

$$y = \frac{z(a_2c_1 - a_1c_2) + (a_2d_1 - a_1d_2)}{(a_1b_2 - a_2b_1)}$$

Thus, for any given $z_0$, $U$, and $V$, we may determine the corresponding $x_0$ and $y_0$, i.e the real-world coordinates.

### 8.6.3 Recovery of the third dimension

The camera model and the inverse perspective transformation which we have just discussed allow us to compute the $x$- and $y$- real-world coordinates corresponding to a given position in the image. However, we must assume that the $z$-coordinate, i.e. the distance from the camera, is known. For the wire-crimping application in which the wires lie on a table at a given and constant height (i.e. at a given $z_0$), this is quite adequate. In general, however, we will *not* know the coordinate of the object in the third dimension and we must recover it somehow. As we have already noted, a significant part of computer vision is concerned with exactly this problem and three techniques for determining depth information will be discussed: one in the next section of this chapter and two in Chapter 9. The purpose of this section is to show how we can compute $z_0$ if we have a second image of the scene, taken from another viewpoint, and if we know the image coordinates of the point of interest (e.g. the wire end) in this image also.

In this instance, we have two camera models and, hence, two inverse perspective transformations. Instead of solving two plane equations simultaneously, we solve four plane equations. In particular, we have:

$$
\begin{aligned}
a_1x + b_1y + c_1z + d_1 &= 0\\
a_2x + b_2y + c_2z + d_2 &= 0\\
p_1x + q_1y + r_1z + s_1 &= 0\\
p_2x + q_2y + r_2z + s_2 &= 0
\end{aligned}
$$

where

$$
\begin{aligned}
a_1 &= C1_{11} - U1C1_{31} & a_2 &= C1_{21} - V1C1_{31}\\
b_1 &= C1_{12} - U1C1_{32} & b_2 &= C1_{22} - V1C1_{32}\\
c_1 &= C1_{13} - U1C1_{33} & c_2 &= C1_{23} - V1C1_{33}\\
d_1 &= C1_{14} - U1C1_{34} & d_2 &= C1_{24} - V1C1_{34}\\
p_1 &= C2_{11} - U2C2_{31} & p_2 &= C2_{21} - V2C2_{31}\\
q_1 &= C2_{12} - U2C2_{32} & q_2 &= C2_{22} - V2C2_{32}\\
r_1 &= C2_{13} - U2C2_{33} & r_2 &= C2_{23} - V2C2_{33}\\
s_1 &= C2_{14} - U2C2_{34} & s_2 &= C2_{24} - V2C2_{34}
\end{aligned}
$$

and $C1_{ij}$ and $C2_{ij}$ are the coefficients of the camera model for the first and second images respectively. Similarly, $U1$, $V1$ and $U2$, $V2$ are the coordinates of the point of interest in the first and second images respectively. Since we now have four equations and three unknowns, the system is overdetermined and we compute a least-square-error solution using the pseudo-inverse technique discussed in Chapter 4.

It should be noted that the key here is not so much the mathematics which allow us to compute $x_0, y_0$, and $z_0$ but, rather, the image analysis by which we identify the corresponding point of interest in the two images. It is this correspondence problem which lies at the heart of most of the difficulties in recovery of depth information. To complete the chapter, the next section describes

a simple, popular, and useful technique for analysing images and computing depth information.

## 8.7 Three-dimensional vision using structured light

Active ranging using structured light is one of the most popular ranging techniques in industrial vision. The essential idea is to illuminate the object in such a way so that

(a) we know the position and direction of the source of illumination;
(b) the point being illuminated is easily identifiable (e.g. we illuminate a very small part of the surface of the object with a dot of light);
(c) we know the position of the sensor (camera) and can compute the direction to the illuminated part of the object surface.

Thus, we can draw two lines, one along the ray of illumination from the light source to the object surface and the other from the position of the sensed illumination on the image through the focal point to the object surface. The object surface is at the intersection of these two lines (see Figure 8.24) and to compute the three-dimensional position of this point on the surface of the object, we just have to compute the point of intersection of these two lines.

The basis of the approach is that it solves, in a simple but contrived way, the correspondence problem to which we alluded in the preceding section. The solution is not without problems, however. Since the approach will yield the range to only one small point on the object's surface, we either have to scan the surface with the dot of light, computing the range at each point, or illuminate more than one point

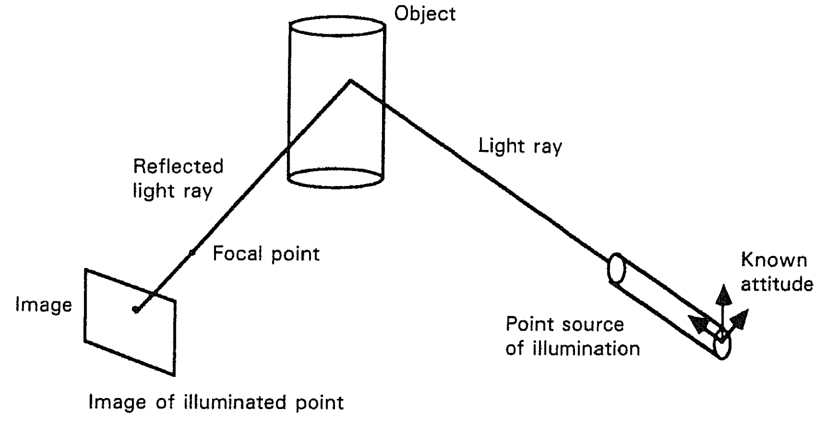


**Figure 8.24** Active triangulation.

at the same time. The former approach is not normally employed, except in specialized laser-scanning applications, since one would have a significant overhead in image acquisition; the latter approach popularly utilizes stripes of light (and is hence referred to as *light striping*) or grids of light to illuminate the object. In these cases, the derivation of range involves the computation of the point of intersection of the plane of light and the line from an image point on the sensed line through the focal point (see Figures 8.25 and 8.26).

If we calibrate the vision system and determine the camera model, then, for any point in the image, we can derive the equation of a single line in the real world. To identify the coordinates of the single point which reflected light causing the imaged point, we need an additional constraint. One such constraint might be the knowledge that the real-world point lies on some plane (which is not coincident with the line of sight). For example, in Section 8.6 where we derived the inverse perspective transformation, we assumed that the point lay on the plane given by $z = z_0$. In the case of light striping, use the same idea and illuminate the object with a single plane of light and, if we know the equation of this plane, then the identification of the three-dimensional world point coordinates simply requires the computation of the intersection of the line of sight (given by the inverse perspective transformation) and this plane. In order to determine the equation of the light plane, one can locate several points on it, identify their three-dimensional coordinates and fit a plane through these points. One simple way of identifying points in the plane of light is to place blocks of different known heights on the work surface in the path of the light beam. Knowing the real $z$-coordinate, the real-world $x$- and $y$-coordinates of points on the resulting stripe are then computed by imaging the stripe and applying the inverse perspective transformation of the camera to the measured points.

Having identified a number ($M$, say) of points on the plane at several different heights, one can use these $x$, $y$, and $z$ values to generate a set of simultaneous plane

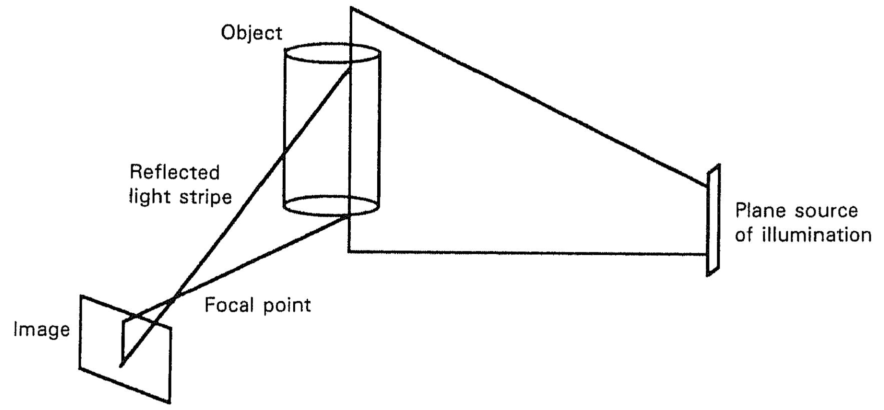


**Figure 8.25** Light striping.

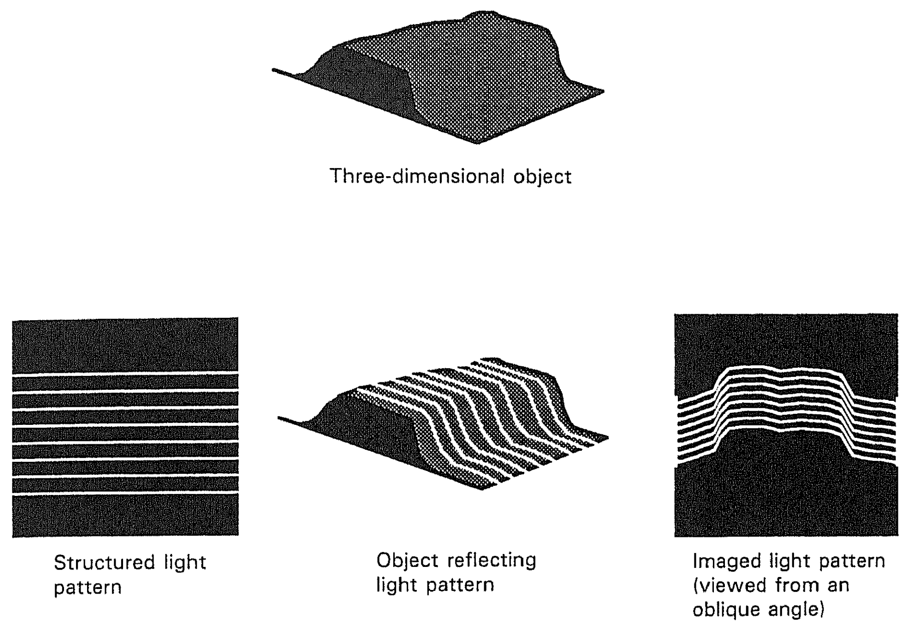

Three-dimensional object
Structured light pattern
Object reflecting light pattern
Imaged light pattern (viewed from an oblique angle)

**Figure 8.26** 'Structured light'.

equations:

$$a_1x_i + a_2y_i + a_3z_i + a_4 = 0, \quad i = 1 \dots M$$

and solve them using the pseudo-inverse method. Unfortunately, this equation has a degenerate solution in which all the coefficients are zero. To avoid this possibility, we can reformulate (1) (from Bolles *et al.*, 1981) by dividing across by $a_3$ and letting:

$$\frac{a_1}{a_3} = b_1$$

$$\frac{a_2}{a_3} = b_2$$

$$\frac{a_4}{a_3} = b_3$$

Thus:

$$b_1x_i + b_2y_i + z_i + b_3 = 0$$

and hence:

$$b_1x_i + b_2y_i + b_3 = z_i$$

A least-square-error solution to this set of equations, written in matrix form as:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ \vdots & & \\ x_M & y_M & 1 \end{bmatrix} * \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} -z_1 \\ -z_2 \\ -z_3 \\ \vdots \\ -z_M \end{bmatrix} \quad M > 3$$

can now be generated using the pseudo-inverse method.

The only restriction in this case is that the plane of light cannot be normal to the $Z$-axis since an equation of this form cannot be used to represent such a plane (i.e. $Z$ cannot be constant).

The equation of the plane of light is thus:

$$b_1x_i + b_2y_i + z_i + b_3 = 0$$

and the three-dimensional position of a point in an imaged light stripe can be found by solving the set of simultaneous equations given by the two plane equations

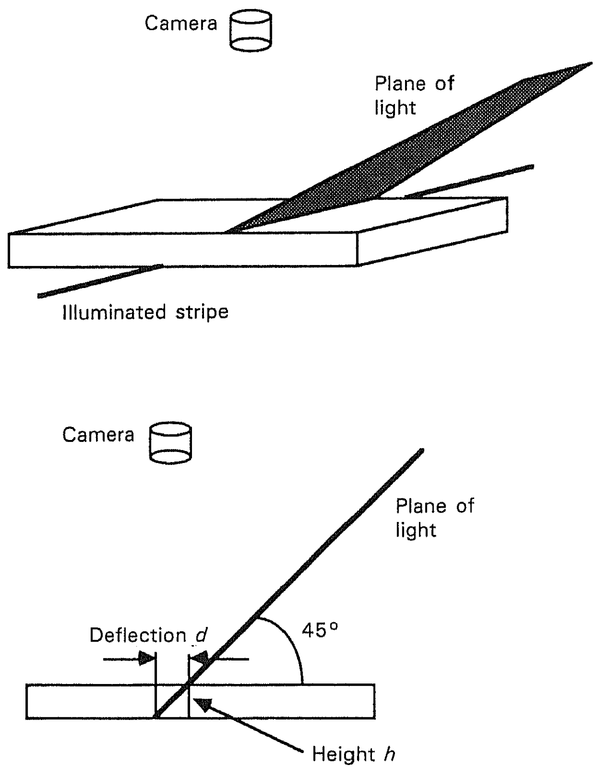


**Figure 8.27** Height measurement using light stripes.

provided by the inverse perspective transformation:

$$x(c_{11} - uc_{31}) + y(c_{12} - uc_{32}) + z(c_{13} - uc_{33}) = uc_{34} - c_{14}$$
$$x(c_{21} - vc_{31}) + y(c_{22} - vc_{32}) + z(c_{23} - vc_{33}) = vc_{34} - c_{24}$$

and the light plane:

$$b_1x + b_2y + z = -b_3$$

The plane of light can be generated either using a laser scanner or it can be generated by projecting a collimated light source through a slit. The advantage of using a laser is that it can illuminate the object in the presence of ambient lighting, e.g. by using an infra-red laser and an infra-red sensitive sensor (CCD sensor), while the slit projection approach will typically require a somewhat darkened environment or an extremely bright light source. Furthermore, this approach suffers from a problem common to all so-called triangulation systems of this type: that only surface points which are visible from both illumination source and sensor can be used to yield range measurements. Hence, hidden parts in concavities will cause some problems.

As a final note, it is worth remarking that this structured light approach is quite general in the sense that it allows you to generate all three real-world coordinates for points on an imaged light stripe. If you are only interested in deriving the height of the object rather than its range, then you can adopt a simpler approach. Consider a plane of light which is incident to the work surface at an angle of 45° (see Figure 8.27). An object on the work surface in the path of the beam will cause the illuminated stripe to be deflected by an amount which is proportional to the height of the block. In fact, for the example shown, the deflection will be equivalent to the height of the block (in an image frame of reference). Thus, to measure the height you merely need to calibrate the system by computing the relationship between a deflection $d$ and a height $h$ in the real world (using a block of known height) and subsequently measure deflections.

## Exercises

1. Describe the use of transform equations in robot task specification, illustrating your answer with at least one example.
2. What is meant by the camera model and the inverse perspective transformation? How do these transformations relate to the transform equations used in the robot task specification?
3. Cylindrical steel ingots, held in a clamp after they have been cut from a bar, require chamfering (i.e. trimming the edge of the cut) to minimize the possibility of jamming when they are introduced into a cast. This can be accomplished by a robot with a high-speed rotating

grinding wheel mounted on the end effector. Identify a sequence of end-effector movements which will effect this chamfering task and generate a complete task specification by:

(i) identifying the appropriate coordinate frames for each distinct object/end-effector position;
(ii) specifying the task transform equations; and
(iii) solving the task transform equations to achieve an explicit sequence of movements for the end effector.

Each coordinate frame specified in (i) above should be explicitly defined and you should use figures where appropriate.

How would you exploit CAD (computer aided design) information regarding the clamp position and the ingot diameter?

4. In some industrial quality control tasks, the objects to be checked and assured are three-dimensional and all the visible surfaces must be viewed and inspected. In what circumstances would it be useful to deploy an articulated robot-mounted camera system rather than several distinct cameras?

Using homogeneous transformations and task transform equations to specify object position and orientation, describe with the aid of diagrams how one would configure a robot program to inspect all five visible faces of a cubic object using a camera mounted on the end effector of the robot.

## References and further reading

Adams, R. 1983 'Welding apparatus with vision correction', *Robotics age*, Nov/Dec, pp. 43–6.

Adorni, G. and Di Manzo, M. 1980 *A Natural Language as a Means of Communications between Men and Robots*, Internal Report, Institute of Electrotechnics, University of Genoa, 1980.

Agin, G. 1979 *Real-time Control of a Robot with a Mobile Camera*, SRI International, Technical Report No. 179.

Agin, G.J. 1985 *Calibration and Use of a Light Stripe Range Sensor Mounted on the Hand of a Robot*, CMU-RI-TR-20, The Robotics Institute, Carnegie-Mellon University.

Ayache, N., Faverjon, B., Boissonnat, J.D. and Bollack, B. 1984 'Manipulation Automatique de Pieces Industrielles en Vrac Planaire', *Proceedings of the First Image Symposium, CESTA, Biarritz*, pp. 869–75.

Bogaert, M. and Ledoux, O. 1983 '3-D perception in industrial environment', *Proceedings of SPIE*, Vol. 449, pp. 373–80.

Bolles, R.C. 1981 *Three-Dimensional Locating of Industrial Parts*, SRI International, Technical Note No. 234.

Bolles, R.C., Kremers, J.H. and Cain, R.A. 1981 *A Simple Sensor to Gather 3-D Data*, SRI International, Technical Note No. 249.

Bonner, S. and Shin, K.G. 1982 'A comparative study of robot languages', *Computer*, Vol. 15, No. 12, pp. 82–96.

Brooks, R.A. 1983 'Planning collision-free motions for pick-and-place operations', *The International Journal of Robotics Research*, Vol. 2, No. 4, pp. 19–44.

Chiang, M.C., Tio, J.B.K. and Hall, E.L. 1983 'Robot vision using a projection method, *Proceedings of SPIE*, Vol. 449, pp. 74–81.

Drezner, Z. and Nof, S.Y. 1984 'On optimizing bin picking and insertion plans for assembly robots', *IIE Transactions*, Vol. 16, No. 3, pp. 262–70.

El-Hakim, S.F. 1985 'A photogrammetric vision system for robots', *Photogrammetric Engineering and Remote Sensing*, Vol. 51, No. 5, pp. 545–52.

Goldman, R. 1982 *Design of an Interactive Manipulator Programming Environment*, Department of Computer Science, Stanford University, Stanford, STAN-CS-82-955.

Grossman, D.D. 1977 *Programming of a Computer Controlled Industrial Manipulator by Guiding through the Motions*, IBM Research Report RC6393, IBM T.J. Watson Research Centre, Yorktown Heights, N.Y.

Guo, H-L., Yachida, M. and Tsuji, S. 1986 'Three dimensional measurement of many line-like objects', *Advanced Robotics*, Vol. 1, No. 2, pp. 117–30.

Hall, E.L., Tio, J.B.K., McPherson, C.A. and Sadjadi, F. 1982 'Measuring curved surfaces for robot vision', *Computer*, Vol. 15, No. 12, pp. 42–54.

Jarvis, R.A. 1983 'A perspective on range finding techniques for computer vision', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-5, No. 2, pp. 122–39.

Lavin, M.A. and Lieberman, L.I. 1982 'AML/V: An industrial machine vision programming system', *The International Journal of Robotics Research*, Vol. 1, No. 3, pp. 42–56.

Lozano-Perez, T. 1982 *Robot Programming*, MIT AI Memo No. 698.

Luh, J.Y.S. and Klaasen, J.A. 1983 'A real-time 3-D multi-camera vision system', *Proceedings of SPIE*, Vol. 449, pp. 400–8.

Luh, J.Y.S. and Klaasen, J.A. 1985 'A three-dimensional vision by off-shelf system with multi-cameras', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-7, No. 1, pp. 35–45.

Mujtaba, M. 1982 'Motion sequencing of manipulators', Ph.D. Thesis, Stanford University, Report No. STAN-CS-82-917.

Mujtaba, M. and Goldman, R. 1979 *The AL User's Manual*, STAN-CS-79-718, Stanford University.

Nagel, R.N. 1984 'Robots: Not yet smart enough', *IEEE Spectrum*, Vol. 20, No. 5, pp. 78–83.

Paul, R. 1979 'Robots, models, and automation', *Computer*, July, pp. 19–27.

Paul, R. 1981 *Robot Manipulators: Mathematics, Programming, and Control*, MIT Press, Cambridge, Massachusetts, 1981.

Summers, P.D. and Grossman, D.D. 1984 'XPROBE: An experimental system for programming robots by example', *The International Journal of Robotics Research*, Vol. 3, No. 1, pp. 25–39.

Taylor, R.H. 1983 *An Integrated Robot System Architecture*, IBM Research Report.

Taylor, R.H., Summers, P.D. and Meyer, J.M. 1982 'AML: A manufacturing language', *The International Journal of Robotics Research*, Vol. 1, No. 3, pp. 19–41.

Tsai, R.Y. 1987 'A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses', *IEEE Journal of Robotics and Automation*, Vol. RA-3, No. 4, pp. 323–44.

Vernon, D. 1985 'A hierarchically-organized robot vision system', *Proceedings of AI EUROPA*, Wiesbaden, West Germany.

Volz, R.A., Mudge, T.N. and Gal, D.A. 1983 *Using Ada as a Programming Language for Robot-Based Manufacturing Cells*, RSD-TR-15-83. Centre of Robotics and Integrated Manufacturing, Robot Systems Division, College of Engineering, University of Michigan.

Yachida, M., Tsuji, S. and Huang, X. 1982 'Wiresight – a computer vision system for 3-D measurement and recognition of flexible wire using cross stripe light', *Proceedings of the 6th International Conference on Pattern Recognition*, Vol. 1, pp. 220–2.

# 9 Introduction to image understanding

## 9.1 *Representations and information processing: from images to object models*

This book began by borrowing a phrase from David Marr and defining computer vision as the endeavour to 'say what is where in the world by looking' by the automatic processing and analysis of images by computer. We immediately distinguished between industrial machine vision and image understanding, identifying the former as the heavily engineered pragmatic application of a small sub-set of the broad spectrum of imaging techniques in quite restricted, and often two-dimensional, domains. Image understanding, on the other hand, addresses general three-dimensional environments where one lifts the restrictions on the possible organization of the visual domain. Thus, image understanding must take into consideration the considerable loss of information which arises when a three-dimensional world is imaged and represented by two-dimensional digital images. In particular, it must address the recovery of range or depth information. We noted that a considerable amount of effort is expended in accomplishing this. Again, approaches which are associated with image understanding endeavour to avoid intrusive sensing techniques, i.e. imaging systems which depend on the transmission of appropriate signals (infra-red, laser, or ultrasonic beams; or grids of structured light) to facilitate the process. Instead, a more passive approach is adopted, using whatever ambient information exists, in an *anthropomorphic* manner.

There is, however, more to image understanding than just the recovery of depth information. If we are going to be able to identify the structure of the environment, we need to do more than develop a three-dimensional range map since this is still an image and the information which we need is still *implicit*. We require an *explicit* representation of the structure of the world we are imaging. Hence, we still need the process of segmentation and object recognition that we dealt with in the preceding chapters. Our difficulty is that now the data we are dealing with is much more complex than before and the simplistic image segmentation, template