A Tutorial on Visual Servo Control

Seth Hutchinson, Greg Hager and Peter Corke

Research Report YALEU/DCS/RR-1068
March 1995

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

A Tutorial on Visual Servo Control

Seth Hutchinson
Department of Electrical and Computer Engineering
The Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana-Champaign
405 N. Mathews Avenue
Urbana, IL 61801
Email: seth@uiuc.edu

Greg Hager
Department of Computer Science
Yale University
New Haven, CT 06520-8285
Phone: 203 432-6432
Email: hager@cs.yale.edu

Peter Corke
CSIRO Division of Manufacturing Technology
P.O. Box 883,
Kenmore. Australia, 4069.
pic@brb.dmt.csiro.au

March 22, 1995

Abstract

This paper provides a tutorial introduction to visual servo control of robotic manipulators. Since the topic spans many disciplines our goal is limited to providing a basic conceptual framework. We begin by reviewing the prerequisite topics from robotics and computer vision, including a brief review of coordinate transformations, velocity representation, and a description of the geometric aspects of the image formation process. We then present a taxonomy of visual servo control systems. The two major classes of systems, position-based and image-based systems, are then discussed. Since any visual servo system must be capable of tracking image features in a sequence of images, we include an overview of feature-based and correlation-based methods for tracking. We conclude the tutorial with a number of observations on the current directions of the research field of visual servo control.

1 Introduction

Today there are over 800,000 robots in the world, mostly working in factory environments. This population continues to grow, but robots are excluded from many application areas where the work enviroment and object placement cannot be accurately controlled. This limitation is due to the inherent lack of sensory capability in contempory commercial robot systems. It has long been recognized that sensor integration is fundamental to increasing the versatility and application domain of robots but to date this has not proven cost effective for the bulk of robotic applications which are in manufacturing. The 'new frontier' of robotics, which is operation in the everyday world, provides new impetus for this research. Unlike the manufacturing application, it will not be cost effective to re-engineer 'our world' to suit the robot.

Vision is a useful robotic sensor since it mimics the human sense of vision and allows for non-contact measurement of the environment. Since the seminal work of Shirai and Inoue [1] (who describe how a visual feedback loop can be used to correct the position of a robot to increase task accuracy), considerable effort has been devoted to the visual control of robot manipulators. Robot controllers with fully integrated vision systems are now available from a number of vendors. Typically visual sensing and manipulation are combined in an open-loop fashion, 'looking' then 'moving'. The accuracy of the resulting operation depends directly on the accuracy of the visual sensor and the robot end-effector.

An alternative to increasing the accuracy of these subsystems is to use a visual-feedback control loop which will increase the overall accuracy of the system — a principal concern in any application. Taken to the extreme, machine vision can provide closed-loop position control for a robot end-effector — this is referred to as *visual servoing.* This term appears to have been first introduced by Hill and Park [2] in 1979 to distinguish their approach from earlier 'blocks world' experiments where the system alternated between picture taking and moving. Prior to the introduction of this term, the less specific term *visual feedback* was generally used. For the purposes of this article, the task in visual servoing is to use visual information to control the *pose* of the robot's end-effector relative to a target object or a set of target features.

Since the first visual servoing systems were reported in the early 1980s, progress in visual control of robots has been fairly slow but the last few years have seen a marked increase in published research. This has been fueled by personal computing power crossing the threshold which allows analysis of scenes at a sufficient rate to 'servo' a robot manipulator. Prior to this, researchers required specialized and expensive pipelined pixel processing hardware. Applications that have been proposed or prototyped span manufacturing (grasping objects on conveyor belts and part mating), teleoperation, missile tracking cameras and fruit picking as well as robotic ping-pong, juggling, balancing, car steering and even aircraft landing. A comprehensive review of the literature in this field, as well the history and applications reported to date, is given by Corke [3] and includes a large bibliography.

Visual servoing is the fusion of results from many elemental areas including high-speed image processing, kinematics, dynamics, control theory, and real-time computing. It has much in common with research into *active vision* and *structure from motion*, but is quite different to the often described use of vision in hierarchical task-level robot control systems. Many of the control and vision

problems are similar to those encountered by active vision researchers who are building 'robotic heads'. However the task in visual servoing is to control a robot to manipulate its environment using vision as opposed to passively or actively observing it.

Given the current interest in this topic it seems both appropriate and timely to provide a tutorial introduction to this topic. We hope that this tutorial will assist researchers by providing a consistant terminology and nomenclature, and assist others in creating visually servoed systems and gaining an appreciation of possible applications. The growing literature contains solutions and promising approaches to many theoretical and technical problems involved. We have attempted here to present the most significant results in a consistant way in order to present a comprehensive view of the area. Another difficulty we faced was that the topic spans many disciplines. Some issues that arise such as the control problem, which is fundamentally nonlinear and for which there is not complete established theory, and visual recognition, tracking, and reconstruction which are fields unto themselves cannot be adequately addressed in a single article. We have thus concentrated on certain fundamental aspects of the topic, and a large bibliography is provided to assist the reader who seeks greater detail than can be provided here. Our preference is always to present those ideas and techniques which have been found to function well in practice in situations where high control and/or vision performance is not required, and which appear to have some generic applicability. In particular we will describe techniques which can be implemented using a minimal amount of vision hardware, and which make few assumptions about the robotic hardware.

The remainder of this article is structured as follows. Section 2 establishes a consistent nomenclature and reviews the relevant fundamentals of coordinate transformations, pose representation, and image formation. In Section 3, we present a taxonomy of visual servo control systems (adapted from [4]). The two major classes of systems, position-based visual servo systems and image-based visual servo systems, are discussed in Sections 4 and 5 respectively. Since any visual servo system must be capable of tracking image features in a sequence of images, Section 6 describes some approaches to visual tracking that have found wide applicability and can be implemented using a minimum of special-purpose hardware. Finally Section 7 presents a number of observations, and about the current directions of the research field of visual servo control.

2 Background and Definitions

Visual servo control research requires some degree of expertise in several areas, particularly robotics, and computer vision. Therefore, in this section we provide a very brief overview of these subjects, as relevant to visual servo control. We begin by defining the terminology and notation required to represent coordinate transformations and the velocity of a rigid object moving through the workspace (Sections 2.1 and 2.2). Following this, we briefly discuss several issues related to image formation, including the image formation process (Sections 2.3 and 2.4), and possible camera/robot configurations (Section 2.5). The reader who is familiar with these topics may wish to proceed directly to Section 3.

2.1 Coordinate Transformations

In this paper, the task space of the robot, represented by $\mathcal{T}$, is the set of positions and orientations that the robot *tool* can attain. Since the task space is merely the configuration space of the robot tool, the task space is a smooth m-manifold (see, e.g., [5]). If the tool is a single rigid body moving arbitrarily in a three-dimensional workspace, then $\mathcal{T} = \mathrm{SE}^3 = \Re^3 \times \mathrm{SO}^3$, and $m = 6$. In some applications, the task space may be restricted to a subspace of SE^3. For example, for pick and place, we may consider pure translations ($\mathcal{T} = \Re^3$, for which $m = 3$), while for tracking an object and keeping it in view we might consider only rotations ($\mathcal{T} = \mathrm{SO}^3$, for which $m = 3$).

Typically, robotic tasks are specified with respect to one or more coordinate frames. For example, a camera may supply information about the location of an object with respect to a camera frame, while the configuration used to grasp the object may be specified with respect to a coordinate frame at the end-effector of the manipulator. We represent the coordinates of point $\mathbf{P}$ with respect to coordinate frame x by the notation ${}^x\mathbf{P}$. Given two frames, x and y, the rotation matrix that represents the orientation of frame y with respect to frame x is denoted by ${}^x\mathbf{R}_y$. The location of the origin of frame y with respect to frame x is denoted by the vector ${}^x\mathbf{t}_y$. Together, the position and orientation of a frame are referred to as its *pose*, which we denote by a pair ${}^x\mathbf{x}_y = ({}^x\mathbf{R}_y, {}^x\mathbf{t}_y)$. If x is not specified, the world coordinate frame is assumed.

If we are given ${}^y\mathbf{P}$ (the coordinates of point $\mathbf{P}$ relative to frame y), and ${}^x\mathbf{x}_y = ({}^x\mathbf{R}_y, {}^x\mathbf{t}_y)$, we can obtain the coordinates of $\mathbf{P}$ with respect to frame x by the coordinate transformation

$$\begin{aligned} {}^x\mathbf{P} &= {}^x\mathbf{R}_y {}^y\mathbf{P} + {}^x\mathbf{t}_y & (1) \\ &= {}^x\mathbf{x}_y \circ {}^y\mathbf{P}. & (2) \end{aligned}$$

Often, we must compose multiple poses to obtain the desired coordinates. For example, suppose that we are given poses ${}^x\mathbf{x}_y$ and ${}^y\mathbf{x}_z$. If we are given ${}^z\mathbf{P}$ and wish to compute ${}^x\mathbf{P}$, we may use the composition of transformations

$$\begin{aligned} {}^x\mathbf{P} &= {}^x\mathbf{x}_y \circ {}^y\mathbf{P} & (3) \\ &= {}^x\mathbf{x}_y \circ {}^y\mathbf{x}_z \circ {}^z\mathbf{P} & (4) \\ &= {}^x\mathbf{x}_z \circ {}^z\mathbf{P} & (5) \end{aligned}$$

where

$${}^x\mathbf{x}_z = ({}^x\mathbf{R}_y {}^y\mathbf{R}_z, {}^x\mathbf{R}_y {}^y\mathbf{t}_z + {}^x\mathbf{t}_y) \quad (6)$$

Thus, we will represent the the composition of two poses by ${}^x\mathbf{x}_z = {}^x\mathbf{x}_y \circ {}^y\mathbf{x}_z$. We note that the operator $\circ$ is used to represent both the coordinate transformation of a single point and the composition of two coordinate transformations. The particular meaning should always be clear from the context.

In much of the robotics literature, poses are represented by homogeneous transformation matrices, which are of the form

$$ {}^{x}\mathbf{T}_{y} = \begin{bmatrix} {}^{x}\mathbf{R}_{y} & {}^{x}\mathbf{t}_{y} \\ \mathbf{0} & 1 \end{bmatrix}. \tag{7} $$

To simplify notation throughout the paper, we will represent poses and coordinate transformations as defined in (1). Some coordinate frames that will be needed frequently are referred to by the following superscripts/subscripts:

e	The coordinate frame attached to the robot end effector
0	The base frame for the robot
c	The camera coordinate frame

When $\mathcal{T} = \mathrm{SE}^3$, we will use the notation $\mathbf{x}_e \in \mathcal{T}$ to represent the pose of the end-effector coordinate frame relative to the world frame. In this case, we often prefer to parameterize a pose using a translation vector and three angles, (e.g., roll, pitch and yaw [6]). Although such parameterizations are inherently local, it is often convenient to represent a pose by a vector $\mathbf{r} \in \Re^6$, rather than by $\mathbf{x}_e \in \mathcal{T}$. This notation can easily be adapted to the case where $\mathcal{T} \subseteq \mathrm{SE}^3$. For example, when $\mathcal{T} = \Re^3$, we will parameterize the task space by $\mathbf{r} = [x, y, z]^T$. In the sequel, to maintain generality we will assume that $\mathbf{r} \in \Re^m$, unless we are considering a specific task.

2.2 The Velocity of a Rigid Object

In visual servo applications, we are often interested the relationship between the velocity of some object in the workspace (e.g., the manipulator end-effector) and the corresponding changes that occur in the observed image of the workspace. In this section, we briefly introduce notation to represent velocities of objects in the workspace.

Consider the robot end-effector moving in a workspace with $\mathcal{T} \subseteq \mathrm{SE}^3$. In base coordinates, the motion is described by an angular velocity $\Omega(t) = [\omega_x(t), \omega_y(t), \omega_z(t)]^T$ and a translational velocity $\mathbf{T}(t) = [T_x(t), T_y(t), T_z(t)]^T$. The rotation acts about a point which, unless otherwise indicated, we take to be the origin of the base coordinate system. Let $\mathbf{P}$ be a point that is rigidly attached to the end-effector, with base frame coordinates $[x, y, z]^T$. The derivatives of the coordinates of $\mathbf{P}$ with respect to base coordinates are given by

$$ \dot{x} = z\omega_y - y\omega_z + T_x \tag{8} $$
$$ \dot{y} = x\omega_z - z\omega_x + T_y \tag{9} $$
$$ \dot{z} = y\omega_x - x\omega_y + T_z \tag{10} $$

which can be written in vector notation as

$$ \dot{\mathbf{P}} = \Omega \times \mathbf{P} + \mathbf{T}. \tag{11} $$

This can be written concisely in matrix form by noting that the cross product can be represented in terms of the *skew-symmetric matrix*

$$\mathrm{sk}(\mathbf{P}) = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}$$

allowing us to write

$$\dot{\mathbf{P}} = -\mathrm{sk}(P)\Omega + \mathbf{T}. \tag{12}$$

Together, $\mathbf{T}$ and Ω define what is known in the robotics literature as a velocity screw

$$\dot{\mathbf{r}} = \begin{bmatrix} T_x \\ T_y \\ T_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}.$$

Note that $\dot{\mathbf{r}}$ also represents the derivative of $\mathbf{r}$ when the angle parameterization is chosen to be the set of rotations about the coordinate axes (recall that $\mathbf{r}$ is a parameterization of $\mathbf{x}_e$).

Define the 3×6 matrix $A(\mathbf{P}) = [I_3 \mid -\mathrm{sk}(\mathbf{P})]$ where I_3 represents the 3×3 identity matrix. Then (12) can be rewritten in matrix form as

$$\dot{\mathbf{P}} = A(\mathbf{P})\dot{\mathbf{r}} \tag{13}$$

Suppose now that we are given a point expressed in end-effector coordinates, ${}^e\mathbf{P}$. Combining (1) and (13), we have

$$\dot{\mathbf{P}} = A(\mathbf{x}_e \circ {}^e\mathbf{P})\dot{\mathbf{r}} \tag{14}$$

Occasionally, it is useful to transform velocity screws among coordinate frames. For example, suppose that ${}^e\dot{\mathbf{r}} = [{}^e\mathbf{T}; {}^e\Omega]$ is the velocity of the end-effector in end-effector coordinates. Then the equivalent screw in base coordinates is

$$\dot{\mathbf{r}} = \begin{bmatrix} \Omega \\ \mathbf{T} \end{bmatrix} = \begin{bmatrix} \mathbf{R}_e {}^e\Omega \\ \mathbf{R}_e {}^e\mathbf{T} - {}^e\Omega \times \mathbf{t}_e \end{bmatrix}.$$

2.3 Camera Projection Models

To control the robot using information provided by a computer vision system, it is necessary to understand the geometric aspects of the imaging process. Each camera contains a lens that forms a 2D projection of the scene on the image plane where the sensor is located. This projection causes direct depth information to be lost so that each point on the image plane corresponds to a ray in 3D space. Therefore, some additional information is needed to determine the 3D coordinates

1

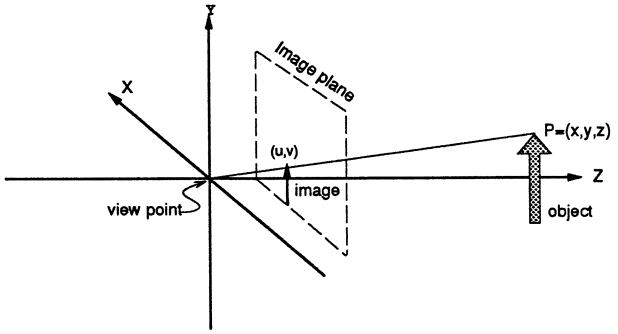

Figure 1: The coordinate frame for the camera/lens system.

corresponding to an image plane point. This information may come from multiple cameras, multiple views with a single camera, or knowledge of the geometric relationship between several feature points on the target. In this section, we describe three projection models that have been widely used to model the image formation process: perspective projection, scaled orthographic projection, and affine projection. Although we briefly describe each of these projection models, throughout the remainder of the tutorial we will assume the use of perspective projection.

For each of the three projection models, we assign the camera coordinate system with the x- and y-axes forming a basis for the image plane, the z-axis perpendicular to the image plane (along the optic axis), and with origin located at distance λ behind the image plane, where λ is the focal length of the camera lens. This is illustrated in Figure 1.

Perspective Projection. Assuming that the projective geometry of the camera is modeled by perspective projection (see, e.g., [7]), a point, ${}^c\mathbf{P} = [x, y, z]^T$, whose coordinates are expressed with respect to the camera coordinate frame, will project onto the image plane with coordinates $\mathbf{p} = [u, v]^T$, given by

$$\pi(x, y, z) = \begin{bmatrix} u \\ v \end{bmatrix} = \frac{\lambda}{z} \begin{bmatrix} x \\ y \end{bmatrix} \tag{15}$$

If the coordinates of $\mathbf{P}$ are expressed relative to coordinate frame x, we must first perform the coordinate transformation ${}^c\mathbf{P} = {}^c\mathbf{x}_x \circ {}^x\mathbf{P}$

Scaled orthographic projection. Perspective projection is a nonlinear mapping from Cartesian to image coordinates. In many cases, it is possible to approximate this mapping by the linear scaled orthographic projection. Under this model, image coordinates for point ${}^c\mathbf{P}$ are given by

$$\begin{bmatrix} u \\ v \end{bmatrix} = s \begin{bmatrix} x \\ y \end{bmatrix} \tag{16}$$

where s is a fixed scale factor.

Orthographic projection models are valid for scenes where the relative depth of the points in the scene is small compared to the distance from the camera to the scene, for example, an airplane flying over the earth, or a camera with a long focal length lens placed several meters from the workspace.

Affine projection. Another linear approximation to perspective projection is known as affine projection. In this case, the image coordinates for the projection of a point ${}^c\mathbf{P}$ are given by

$$\begin{bmatrix} u \\ v \end{bmatrix} = \mathbf{A}\,{}^c\mathbf{P} + \mathbf{c} \tag{17}$$

where $\mathbf{A}$ is an arbitrary 2×3 matrix and $\mathbf{c}$ is an arbitrary 2-vector.

Note that orthographic projection is a special case of affine projection. Affine projection does not correspond to any specific imaging situation. Its primary advantage is that it is an unconstrained linear imaging model. As a result, given a set of corresponding pairs $\{({}^c\mathbf{P}_i, [u_i, v_i]^T)\}$, $\mathbf{A}$ and $\mathbf{c}$ are easily computed using linear regression techniques. Hence, the calibration problem is greatly simplified for this model.

2.4 Image Features and the Image Feature Parameter Space

In the computer vision literature, an *image feature* is any structural feature than can be extracted from an image (e.g., an edge or a corner). Typically, an image feature will correspond to the projection of a physical feature of some object (e.g., the robot tool) on the camera image plane. We define an *image feature parameter* to be any real-valued quantity that can be calculated from one or more image features. Examples include, moments, relationships between regions or vertices, and polygon face areas. Jang [8] provides a formal definition of what we term feature paramters as image functionals. Most commonly the coordinates of a feature point or a region centroid are used. A good feature point is one that can be located unambiguously in different views of the scene, such as a hole in a gasket [9] or a contrived pattern [10, 11]. Image feature parameters that have been used for visual servo control include the image plane coordinates of points in the image [10, 12–17], the distance between two points in the image plane and the orientation of the line connecting those two points [9, 18], perceived edge length [19], the area of a projected surface and the relative areas of two projected surfaces [19], the centroid and higher order moments of a projected surface [19–22], the parameters of lines in the image plane [10], and the parameters of an ellipse in the image plane [10]. In this tutorial we will restrict our attention to point features whose parameters are their image plane coordinates.

In order to perform visual servo control, we must select a set of image feature parameters. Once we have chosen a set of k image feature parameters, we can define an image feature parameter vector $\mathbf{f} = [f_1 \cdots f_k]^T$. Since each f_i is a (possibly bounded) real valued parameter, we have $\mathbf{f} = [f_1 \cdots f_k]^T \in \mathcal{F} \subseteq \Re^k$, where $\mathcal{F}$ represents the *image feature parameter space.*

The mapping from the position and orientation of the end-effector to the corresponding image

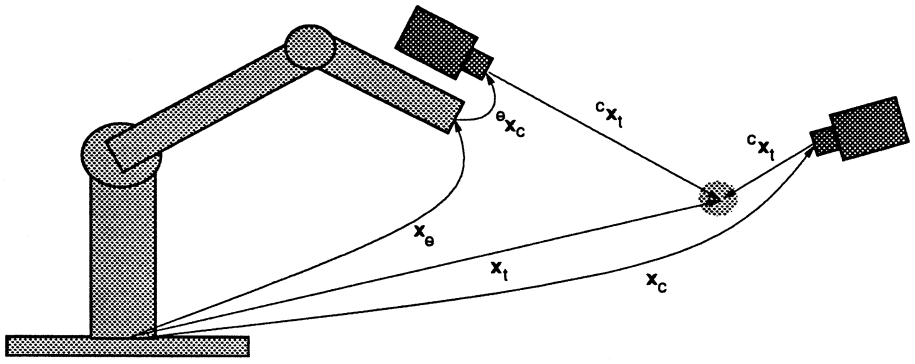

Figure 2: Relevant coordinate frames; world, end-effector, camera and target.

feature parameters can be computed using the projective geometry of the camera. We will denote this mapping by $\mathbf{F}$, where

$$\mathbf{F} : \mathcal{T} \rightarrow \mathcal{F}. \tag{18}$$

For example, if $\mathcal{F} \subseteq \Re^2$ is the space of u, v image plane coordinates for the projection of some point $\mathbf{P}$ onto the image plane, then, assuming perspective projection, $\mathbf{F} = [u, v]^T$, where u and v are given by (15). The exact form of (18) will depend in part on the relative configuration of the camera and end-effector as discussed in the next section.

2.5 Camera Configuration

Visual servo systems typically use one of two camera configurations: end-effector mounted, or fixed in the workspace.

The first, often called an eye-in-hand configuration, has the camera is mounted on the robot's end-effector. Here, there exists a known, often constant, relationship between the pose of the camera(s) and the pose of the end-effector. We represent this relationship by the pose $^{e}\mathbf{x}_{c}$. The pose of the target[1] relative to the camera frame is represented by $^{c}\mathbf{x}_{t}$. The relationship between these poses is shown in Figure 2.

The second configuration has the camera(s) is fixed in the workspace. In this case, the camera(s) are related to the base coordinate system of the robot by $^{0}\mathbf{x}_{c}$ and to the object by $^{c}\mathbf{x}_{t}$. In this case, the camera image of the target is, of course, independent of the robot motion (unless the target is the end-effector itself). A variant of this is for the camera to be agile, mounted on another robot or pan/tilt head in order to observe the visually controlled robot from the best vantage [23].

For either choice of camera configuration, prior to the execution of visual servo tasks, camera calibration must be performed. For the eye-in-hand case, this amounts to determining $^{e}\mathbf{x}_{c}$. For the fixed camera case, calibration is used to determine $^{0}\mathbf{x}_{c}$. Calibration is a long standing research issue in the computer vision community (good solutions to the calibration problem can be found in a number of references, e.g., [24–26]).

[1]The word *target* will be used to refer to the object of interest, that is, the object that will be tracked.

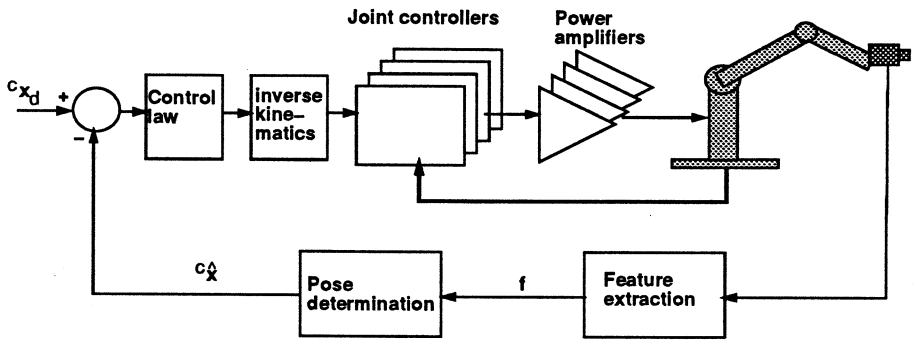

Figure 3: Dynamic position-based look-and-move structure.

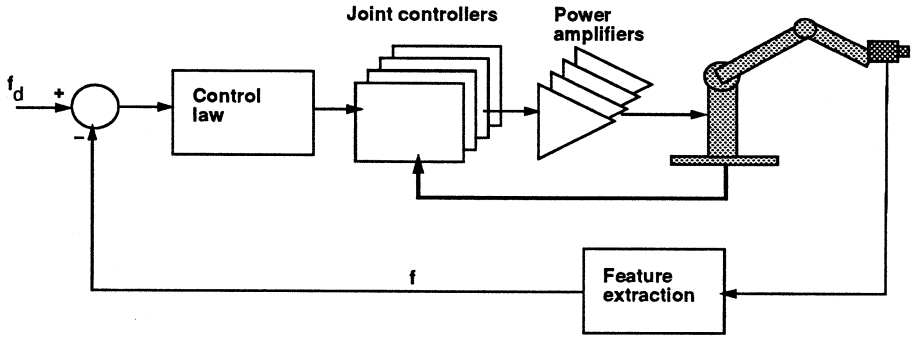

Figure 4: Dynamic image-based look-and-move structure.

3 Servoing Architectures

In 1980, Sanderson and Weiss [4] introduced a taxonomy of visual servo systems, into which all subsequent visual servo systems can be categorized. Their scheme essentially poses two questions:

1. Is the control structure hierarchical, with the vision system providing set-points as input to the robot's joint-level controller, or does the visual controller directly compute the joint-level inputs?
2. Is the error signal defined in 3D (task space) coordinates, or directly in terms of image features?

The resulting taxonomy, thus, has four major categories, which we now describe. These fundamental structures are shown schematically in Figures 3 to 6.

If the control architecture is hierarchical and uses the vision system to provide set-point inputs to the joint-level controller, thus making use of joint feedback to internally stabilize the robot, it is referred to as a *dynamic look-and-move* system. In contrast, *direct visual servo*[2] eliminates

[2]Sanderson and Weiss used the term "visual servo" for this type of system, but since then this term has come to be accepted as a generic description for any type of visual control of a robotic system. Here we use the term "direct

the robot controller entirely replacing it with a visual servo controller that directly computes joint inputs, thus using vision alone to stabilize the mechanism.

For several reasons, nearly all implemented systems adopt the dynamic look-and-move approach. First, the relatively low sampling rates available from vision make direct control of a robot end-effector with complex, nonlinear dynamics an extremely challenging control problem. Using internal feedback with a high sampling rate generally presents the visual controller with idealized axis dynamics [27]. Second, many robots already have an interface for accepting Cartesian velocity or incremental position commands. This simplifies the construction of the visual servo system, and also makes the methods more portable. Thirdly, look-and-move separates the kinematic singularities of the mechanism from the visual controller, allowing the robot to be considered as an ideal Cartesian motion device. Since many resolved rate [28] controllers have specialized mechanisms for dealing with kinematic singularities [29], the system design is again greatly simplified. In this article, we will utilize the look-and-move model exclusively.

The second major classification of systems distinguishes *position-based* control from *image-based* control. In *position-based* control, features are extracted from the image and used in conjunction with a geometric model of the target and the known camera model to estimate the pose of the target with respect to the camera. Feedback is computed by reducing errors in estimated pose space. In *image-based* servoing, control values are computed on the basis of image features directly. The image-based approach may reduce computational delay, eliminate the necessity for image interpretation and eliminate errors due to sensor modeling and camera calibration. However it does present a significant challenge to controller design since the plant is non-linear and highly coupled.

In addition to these considerations, we distinguish between systems which only observe the target object and those which observe both the target object and the robot end-effector. The former are referred to as *endpoint open-loop* (EOL) systems, and the latter as *endpoint closed-loop* (ECL) systems. The primary difference is that EOL system must rely on an explicit hand-eye calibration when translating a task specification into a visual servoing algorithm. Hence, the positioning accuracy of EOL systems depends directly on the accuracy of the hand-eye calibration. Conversely, systems that observe the end-effector as well as target features can perform with accuracy that is independent of hand-eye calibration error [30–32]. Note also that ECL systems can easily deal with tasks that involve the positioning of objects within the end-effector, whereas EOL systems must use an inferred object location.

From a theoretical perspective, it would appear that ECL systems would always be preferable to EOL systems. However, since ECL systems must track the end-effector as well as the target object, the implementation of an ECL controller often requires solution of a more demanding vision problem.

---

visual servo" to avoid confusion.

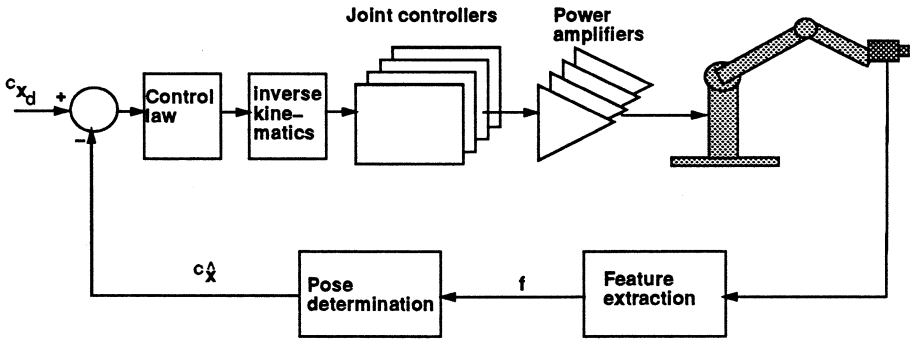

Figure 5: Position-based visual servo (PBVS) structure as per Weiss.

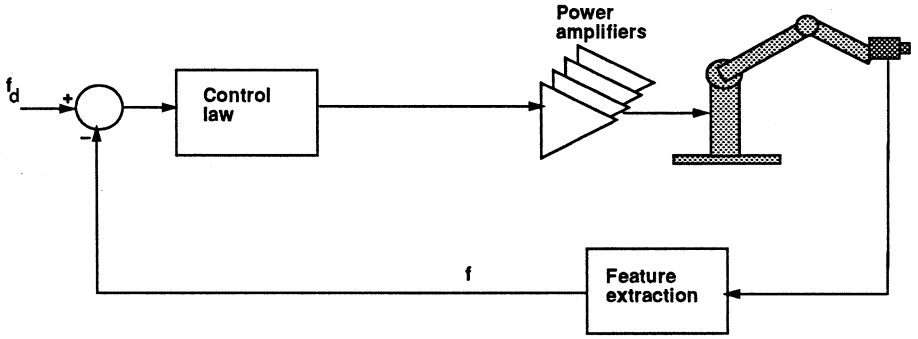

Figure 6: Image-based visual servo (IBVS) structure as per Weiss.

4 Position-Based Visual Servo Control

We begin our discussion of visual servoing methods with position-based visual servoing. As described in the previous section, in position-based visual servoing features are extracted from the image and used to estimate the pose of the target with respect to the camera. Using these values, an error between the current and the desired pose of the robot is defined in the task space. In this way, position-based control neatly separates the control issues, namely the the computation of the feedback signal, from the estimation problems involved in computing position or pose from visual data.

We now formalize the notion of a positioning task as follows:

Definition 4.1 A *positioning task* is represented by a function $\mathbf{E} : \mathcal{T} \rightarrow \Re^m$. This function is referred to as the *kinematic error function.* A positioning task is *fulfilled* with the end-effector in pose $\mathbf{x}_e$ if $\mathbf{E}(\mathbf{x}_e) = \mathbf{0}$.

If we consider a general pose $\mathbf{x}_e$ for which the task is fulfilled, the error function will constrain some number, $d \leq m$, degrees of freedom of the manipulator. The value d will be referred to as the *degree* of the constraint. As noted by Espiau et al. [10,33], the kinematic error function can be

thought of as representing a *virtual kinematic constraint* between the end-effector and the target.

Once a suitable kinematic error function has been defined and the parameters of the functions are instantiated from visual data, a regulator is defined which reduces the estimated value of the kinematic error function to zero. This regulator produces at every time instant a desired end-effector velocity screw $\mathbf{u} \in \Re^6$ which is sent to the robot control subsystem. For the purposes of this section, we use simple proportional control methods for linear and linearized systems to compute $\mathbf{u}$ [34]. These methods are illustrated below, and are discussed in more detail in Section 5.

We now present examples of positioning tasks for end-effector and fixed cameras in both ECL and EOL configurations. In Section 4.1, several examples of positioning tasks based on directly observable features are presented. Following that, Section 4.2, describes positioning tasks based on target pose estimates. Finally, in Section 4.3, we briefly describe how point position and object pose can be computed using visual information.

4.1 Feature-Based Motions

We begin by considering a positioning task in which some point on the robot end-effector, ${}^e\mathbf{P}$, is to be brought to a fixed stationing point, $\mathbf{S}$, visible in the scene. We refer to this as *point-to-point positioning*. In this case, the kinematic error function may be defined in base coordinates as

$$\mathbf{E}_{pp}(\mathbf{x}_e; \mathbf{S}, {}^e\mathbf{P}) = \mathbf{x}_e \circ {}^e\mathbf{P} - \mathbf{S}. \tag{19}$$

Here, as in the sequel, the arguments of the error function after the semicolon denote parameters defining the positioning task.

$\mathbf{E}_{pp}$ defines a three degree of freedom kinematic constraint on the robot end-effector position. If the robot workspace is restricted to be $\mathcal{T} = \Re^3$, this task can be thought of as a rigid link that fully constrains the pose of the end-effector relative to the target. When $\mathcal{T} \subseteq \mathrm{SE}^3$, the constraint defines a *virtual spherical joint* between the object and the robot end-effector.

Let $\mathcal{T} = \Re^3$. We first consider the case in which one or more cameras calibrated to the robot base frame furnish an estimate, ${}^c\widehat{\mathbf{S}}$, of the stationing point coordinates with respect to a camera coordinate frame. Using the estimate of the camera pose in base coordinates, $\hat{\mathbf{x}}_c$, from offline calibration and (1), we have $\widehat{\mathbf{S}} = \hat{\mathbf{x}}_c \circ {}^c\widehat{\mathbf{S}}$.

Since $\mathcal{T} = \Re^3$, the control input to be computed is the desired robot translational velocity which we denote by $\mathbf{u}_3$ to distinguish it from the more general end-effector screw. Since (19) is linear in $\mathbf{x}_e$, it is well known that in the absence of outside disturbances, the proportional control law

$$\mathbf{u}_3 = -kE_{pp}(\hat{\mathbf{x}}_e; \hat{\mathbf{x}}_c \circ {}^c\widehat{\mathbf{S}}, {}^e\mathbf{P}). \tag{20}$$

will drive the system to an equilibrium state in which the estimated value of the error function is zero [34]. The value $k > 0$ is a proportional feedback gain. Note that we have written $\hat{\mathbf{x}}_e$ in the feedback law to emphasize the fact that this value is also subject to errors.

The expression (20) is equivalent to open-loop positioning of the manipulator based on vision-based estimates of geometry. Errors in $\hat{\mathbf{x}}_e$, $\hat{\mathbf{x}}_c$ or ${}^c\widehat{\mathbf{S}}$ (robot kinematics, camera calibration and visual reconstruction respectively) will lead to positioning errors of the end-effector.

Now, consider the situation when the cameras are mounted on the robot and calibrated to the end-effector. In this case, we can express (19) in end-effector coordinates:

$$^{e}\mathbf{E}_{pp}(\mathbf{x}_e; \mathbf{S}, {}^{e}\mathbf{P}) = {}^{e}\mathbf{P} - {}^{e}\mathbf{x}_0 \circ \mathbf{S}. \tag{21}$$

The camera(s) furnish an estimate of the stationing point, $^{c}\widehat{\mathbf{S}}$, which can be combined with information from the camera calibration and robot kinematics to produce $\widehat{\mathbf{S}} = \hat{\mathbf{x}}_e \circ {}^{e}\hat{\mathbf{x}}_c \circ {}^{c}\widehat{\mathbf{S}}$. We now compute

$$^{e}\mathbf{u}_3 = -k\ {}^{e}\mathbf{E}_{pp}(\hat{\mathbf{x}}_e; \hat{\mathbf{x}}_e \circ {}^{e}\hat{\mathbf{x}}_c \circ {}^{c}\widehat{\mathbf{S}}, {}^{e}\mathbf{P}) = -k({}^{e}\mathbf{P} - {}^{e}\mathbf{x}_0 \circ {}^{0}\hat{\mathbf{x}}_e \circ {}^{e}\hat{\mathbf{x}}_c \circ {}^{c}\widehat{\mathbf{S}}) = -k({}^{e}\mathbf{P} - {}^{e}\hat{\mathbf{x}}_c \circ {}^{c}\widehat{\mathbf{S}}) \tag{22}$$

Notice that the terms involving $\hat{\mathbf{x}}_e$ have dropped out. Thus (22) is not only simpler, but positioning accuracy is also independent of the accuracy of the robot kinematics.

The above formulations presumed an EOL system. For an ECL system we suppose that we can also directly observe $^{e}\mathbf{P}$ and estimate its coordinates. In this case, (20) and (22) can be written:

$$\mathbf{u}_3 = -k\ E_{pp}(\hat{\mathbf{x}}_e; \hat{\mathbf{x}}_c \circ {}^{c}\widehat{\mathbf{S}}, {}^{e}\hat{\mathbf{x}}_c \circ {}^{c}\widehat{\mathbf{P}}) = -k\ \hat{\mathbf{x}}_c \circ ({}^{c}\widehat{\mathbf{S}} - {}^{c}\widehat{\mathbf{P}}) \tag{23}$$

$$^{e}\mathbf{u}_3 = -k\,{}^{e}E_{pp}(\hat{\mathbf{x}}_e; \hat{\mathbf{x}}_c \circ {}^{c}\widehat{\mathbf{S}}, {}^{e}\hat{\mathbf{x}}_c \circ {}^{c}\widehat{\mathbf{P}}) = -k\ {}^{e}\hat{\mathbf{x}}_c \circ ({}^{c}\widehat{\mathbf{P}} - {}^{c}\widehat{\mathbf{S}}) \tag{24}$$

respectively. We now see that $\mathbf{u}_3$ (respectively $^{e}\mathbf{u}_3$) does not depend on $\hat{\mathbf{x}}_e$ and is homogeneous in $\hat{\mathbf{x}}_c$ (respectively $^{e}\hat{\mathbf{x}}_c$). Hence, if $^{c}\widehat{\mathbf{S}} = {}^{c}\widehat{\mathbf{P}}$, then $\mathbf{u}_3 = \mathbf{0}$, *independent of errors in the robot kinematics or the camera calibration.* This is an important advantage for systems where a precise camera/end-effector relationship is difficult or impossible to determine offline.

Suppose now that $\mathcal{T} \subseteq \mathrm{SE}^3$. Now the control input is $\mathbf{u} \in \mathbf{R}^6$ which represents a complete velocity screw. The error function only constrains 3 degrees of freedom, so the problem of computing $\mathbf{u}$ from the estimated error is underconstrained. One way of proceeding is as follows. Consider the case of free standing cameras. Then in base coordinates we know that $\dot{\mathbf{P}} = \mathbf{u}_3$. Using (13), we can relate this to the end-effector velocity screw as follows:

$$\dot{\mathbf{P}} = \mathbf{u}_3 = \mathbf{A}(\mathbf{P})\mathbf{u} \tag{25}$$

Unfortunately, $\mathbf{A}$ is not square and therefore can cannot be inverted to solve for $\mathbf{u}$. However, recall that the matrix right inverse for an $m \times n$ matrix $\mathbf{M}$, $n > m$ is defined as $\mathbf{M}^{+} = \mathbf{M}^{T}(\mathbf{M}\mathbf{M}^{T})^{-1}$. The right inverse computes the minimum norm vector which solves original system of equations. Hence, we have

$$\mathbf{u} = \mathbf{A}(\mathbf{P})^{+}\mathbf{u}_3 \tag{26}$$

for free-standing cameras. Similar manipulations yield

$$^{e}\mathbf{u} = \mathbf{A}({}^{e}\mathbf{S})^{+}{}^{e}\mathbf{u}_3 \tag{27}$$

for end-effector mounted cameras.

As a second example of feature-based positioning, consider that some point on the end-effector, $^{e}\mathbf{P}$, is to be brought to the *line joining* two fixed points $\mathbf{S_1}$ and $\mathbf{S_2}$ in the world. Geometrically,

the shortest path for performing this task is to move ${}^e\mathbf{P}$ toward the line joining $\mathbf{S_1}$ and $\mathbf{S_2}$ along the perpendicular to the line. The error function describing this trajectory in base coordinates is:

$$\mathbf{E}_{pl}(\mathbf{x}_e; \mathbf{S_1}, \mathbf{S_2}, {}^e\mathbf{P}) = (\mathbf{S_2} - \mathbf{S_1}) \times ((\mathbf{x}_e \circ {}^e\mathbf{P} - \mathbf{S_1}) \times (\mathbf{S_2} - \mathbf{S_1})).$$

Notice that although $\mathbf{E}$ is a mapping from $\mathcal{T}$ to $\Re^3$, placing a point on a line is a constraint of degree 2. From the geometry of the problem, we see that defining

$$\mathbf{u} = -k\mathbf{A}(\hat{\mathbf{x}}_e \circ {}^e\mathbf{P})^+ \mathbf{E}_{pl}(\hat{\mathbf{x}}_e; \widehat{\mathbf{S_1}}, \widehat{\mathbf{S_2}}, {}^e\mathbf{P})$$

is a proportional feedback law for this problem.

Suppose that now we apply this constraint to two points on the end-effector:

$$\mathbf{E}_{ppl}(\mathbf{x}_e; \mathbf{S_1}, \mathbf{S_2}, {}^e\mathbf{P_1}, {}^e\mathbf{P_2}) = \begin{bmatrix} \mathbf{E}_{pl}(\mathbf{x}_e; \mathbf{S_1}, \mathbf{S_2}, {}^e\mathbf{P_1}) \\ \mathbf{E}_{pl}(\mathbf{x}_e; \mathbf{S_1}, \mathbf{S_2}, {}^e\mathbf{P_2}) \end{bmatrix}$$

$\mathbf{E}_{ppl}$ defines a four degree of freedom positioning constraint which aligns the points on the end-effector with those in target coordinates. The error function is again overparameterized. Geometrically, it is easy to see that one way of computing feedback is to compute a translation, $\mathbf{T}$, which moves ${}^e\mathbf{P_1}$ to the line through $\mathbf{S_1}$ and $\mathbf{S_2}$. Simultaneously, we can choose Ω so as to rotate ${}^e\mathbf{P_2}$ about ${}^e\mathbf{P_1}$ so that the the line through ${}^e\mathbf{P_1}$ and ${}^e\mathbf{P_2}$ becomes parallel to that through $\mathbf{S_1}$ and $\mathbf{S_2}$. This leads to the proportional feedback law:

$$\Omega = -k_1(\mathbf{S_2} - \mathbf{S_1}) \times [R_e({}^e\mathbf{P_2} - {}^e\mathbf{P_1})] \quad (28)$$

$$\mathbf{T} = -k_2(\mathbf{S_2} - \mathbf{S_1}) \times ((\hat{\mathbf{x}}_e \circ {}^e\mathbf{P} - \mathbf{S_1}) \times (\mathbf{S_2} - \mathbf{S_1})) - \Omega \times (\hat{\mathbf{x}}_e \circ {}^e\mathbf{P_1}) \quad (29)$$

Note that we are still free to choose translations along the line joining $\mathbf{S_1}$ and $\mathbf{S_2}$ as well as rotations about it. Full six degree-of-freedom positioning can be attained by enforcing another point-to-line constraint using an additional point on the end-effector and an additional point in the world. See [35] for details.

These formulations can be adjusted for end-effector mounted camera and can be implemented as ECL or EOL systems. We leave these modifications as an exercise for the reader.

4.2 Pose-Based Tasks

In the previous section, positioning was defined in terms of directly observable features. When working with *a priori* known objects, it is possible to recover the *pose* of the object, and to define stationing points in object coordinates.

The methods of the previous section can be easily applied when object pose is available. For example, suppose ${}^t\mathbf{S}$ is an arbitrary stationing point in a target object's coordinate system, and that we can compute ${}^e\hat{\mathbf{x}}_t$ using end-effector mounted cameras. Then using (1) we can compute ${}^e\widehat{\mathbf{S}} = {}^e\hat{\mathbf{x}}_t {}^t\mathbf{S}$. This estimate can be used in any of the end-effector based feedback methods of the previous section in both ECL and EOL configurations. Similar remarks hold for systems utilizing free-standing cameras.

Given object pose, it is possible to directly define manipulator stationing in object coordinates. Let ${}^{e^*}\mathbf{x}_t$ be a desired stationing point for the end-effector, and suppose the system employs free-standing cameras. We can define a positioning error

$$\mathbf{E}_{rp}(\mathbf{x}_e; {}^t\mathbf{x}_{e^*}, \mathbf{x}_t) = {}^e\mathbf{x}_{e^*} = {}^e\mathbf{x}_0 \circ \mathbf{x}_t \circ {}^t\mathbf{x}_{e^*}. \tag{30}$$

(Note that in order for this error function to be in accord with our definition of kinematic error we must select a parameterization of rotations which is **0** when the end-effector is in the desired position.)

Using feature information and the camera calibration, we can directly estimate $\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_c \circ {}^c\hat{\mathbf{x}}_t$. In order to compute a velocity screw, we first note that the rotation matrix ${}^{e^*}\mathbf{R}_e$ can be represented as a rotation through an angle ${}^{e^*}\theta_e$ about an axis defined by a unit vector ${}^{e^*}\mathbf{k}_e$ [6]. Thus, we can define

$$\Omega = k_1 {}^{e^*}\hat{\theta}_e {}^{e^*}\hat{\mathbf{k}}_e \tag{31}$$

$$T = k_2 {}^{e^*}\hat{\mathbf{t}}_e - \mathbf{t}_e \times \Omega \tag{32}$$

where $\mathbf{t}_e$ is the origin of the end-effector frame in base coordinates.

Note that if we can also observe the end-effector and estimate its pose, ${}^c\hat{\mathbf{x}}_e$ we can rewrite (30) as follows:

$${}^e\hat{\mathbf{x}}_{e^*} = {}^e\hat{\mathbf{x}}_c \circ {}^c\hat{\mathbf{x}}_0 \circ {}^0\hat{\mathbf{x}}_c \circ {}^c\hat{\mathbf{x}}_t \circ {}^t\mathbf{x}_{e^*} = {}^e\hat{\mathbf{x}}_c \circ {}^c\hat{\mathbf{x}}_t \circ {}^t\mathbf{x}_{e^*}$$

Once again we see that for an ECL system, both the robot kinematic chain and the camera pose relative to the base coordinate system have dropped out of the error equation. Hence, these factors do not affect the positioning accuracy of the system.

The modifications of pose-based methods to end-effector based systems are completely straightforward and are left for the reader.

4.3 Estimation

Obviously, a key issue in position-based visual servo is the estimation of the quantities used to parameterize the feedback. In this regard, position-based visual servoing is closely related to the problem of recovering scene geometry from one or more camera images. This encompasses problems including structure from motion, exterior orientation, stereo reconstruction, and absolute orientation. A comprehensive discussion of these topics can be found in a recent review article [36].

We divide the estimation problems that arise into single-camera and multiple-camera situations which will be discussed in the following sections.

4.3.1 Single Camera

As noted previously, it follows from (15) that a point in a single camera image corresponds to a line in space. Although it is possible to perform geometric reconstruction using a single moving camera, the equations governing this process are often ill-conditioned, leading to stability problems [36] Better results can be achieved if target features have some internal structure, or the features come from a known object. Below, we briefly describe methods for performing both point estimation and pose estimation with a single camera assuming such information is available.

Single Points Clearly, extra information is needed in order to reconstruct the Cartesian coordinates of a point in space from a single camera projection. In particular, if the feature has a known scale, this information can be used to compute point position. An example of such a feature is a circular opening with known diameter d whose image will be an ellipse. By estimating the parameters of the ellipse in the image, it is possible to compute distance to the hole as well as the orientation of the plane of the hole relative to the camera imaging plane.

Object Pose Accurate object pose estimation is possible if the vision system observes features of a known object, and uses those features to estimate object pose. This approach has been recently demonstrated by Wilson [37] for six DOF control of end-effector pose. A similar approach was recently reported in [38].

Briefly, such an approach proceeds as follows. Let ${}^t\mathbf{P}_1, {}^t\mathbf{P}_2, \ldots {}^t\mathbf{P}_n$ be a set of points expressed in an object coordinate system with unknown pose ${}^c\mathbf{x}_t$ relative to an observing camera. The reconstruction problem is to estimate ${}^c\mathbf{x}_t$ from the image locations of the corresponding observations $\mathbf{p}_1, \mathbf{p}_2, \ldots \mathbf{p}_n$. This is referred to as the *pose estimation* problem in the vision literature. Numerous methods of solution have been proposed and [39] provides a recent review of several techniques. Broadly speaking, solutions divide into analytic solutions and least-squares solutions which employ a variety of simplifications and/or iterative methods. Analytic solutions for three and four points are given by [40–44]. Unique solutions exist for four coplanar, but not collinear, points. Least-squares solutions can be found in [45–51]. Six or more points always yield unique solutions. The camera calibration matrix can be computed from features on the target, then decomposed [49] to yield the target's pose.

The least-squares solution proceeds as follows. Using (15), we can define an objective function of the unknown pose between the camera and the object:

$$O({}^c\mathbf{x}_t) = O({}^c\mathbf{R}_t, {}^c\mathbf{t}_t) = \sum_{i=1}^{n} \|\mathbf{p}_i - \pi({}^c\mathbf{R}_t {}^t\mathbf{P} + {}^c\mathbf{t}_t)\|^2.$$

This is a nonlinear optimization problem which has no known closed-form solution. Instead, iterative optimization techniques are employed. These techniques iteratively refine a nominal value ${}^c\mathbf{x}_t$, (*e.g.* the pose of the object in a previous image), to compute an updated value for the pose parameters. Because of the sensitivity of the reconstruction process to noise, it is often a good idea to incorporate some type of smoothing or averaging of the computed pose parameters, at the

cost of some delay in response to changes in target pose. A particularly elegant formulation of this updating procedure results by application of statistical techniques such as the extended Kalman filter [52]. The reader is referred to [37] for details.

4.3.2 Multiple Cameras

Many systems utilizing position-based control with stereo vision from free-standing cameras have been demonstrated. For example, Allen [53] shows a system which can grasp a toy train using stereo vision. Rizzi [54] demonstrates a system which can bounce a ping-pong ball. All of these systems are EOL. Cipolla [31] describes an ECL system using free-standing stereo cameras. One novel feature of this system is the use of the affine projection model (Section 2.3) for the imaging geometry. This leads to linear calibration and control at the cost of some system performance. The development of a position-based stereo eye-in-hand servoing system has also been reported [55].

Multiple cameras greatly simplify the reconstruction process as illustrated below.

Single Points Let ${}^a\mathbf{x}_{c1}$ represent the location of a camera relative to an arbitrary base coordinate frame a. By inverting this transformation and combining (1) and (15) for a point ${}^a\mathbf{P} = [x, y, z]^T$ we have

$$\mathbf{p}_1 = \begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \frac{\lambda}{\mathbf{z}\,{}^a\mathbf{P} + t_z} \begin{bmatrix} \mathbf{x}\,{}^a\mathbf{P} + t_x \\ \mathbf{y}\,{}^a\mathbf{P} + t_y \end{bmatrix} \tag{33}$$

where $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ are the rows of ${}^{c1}\mathbf{R}_a$ and ${}^{c1}\mathbf{t}_a = [t_x, t_y, t_z]^T$. Multiplying through by the denominator of the right-hand side, we have

$$A_1(\mathbf{p}_1){}^a\mathbf{P} = b_1(\mathbf{p}_1). \tag{34}$$

where

$$A_1(\mathbf{p}_1) = \begin{bmatrix} \lambda\mathbf{x} - u_1\mathbf{z} \\ \lambda\mathbf{y} - v_1\mathbf{z} \end{bmatrix} {}^a\mathbf{P} \quad \text{and} \quad b_1(\mathbf{p}_1) = \begin{bmatrix} t_z u_1 - \lambda t_x \\ t_z v_1 - \lambda t_y \end{bmatrix}.$$

Given a second camera at location ${}^{c2}\mathbf{x}_a$, we can compute $A_2(\mathbf{p}_2)$ and $b_2(\mathbf{p}_2)$ analogously. Stacking these together results in a matrix equation

$$\begin{bmatrix} A_1(\mathbf{p}_1) \\ A_2(\mathbf{p}_2) \end{bmatrix} {}^a\mathbf{P} = \begin{bmatrix} b_1(\mathbf{p}_1) \\ b_2(\mathbf{p}_2) \end{bmatrix}.$$

which is an overdetermined system that can be solved for ${}^a\mathbf{P}$.

Object Pose Given a known object with three or more points in known locations with respect to an object coordinate system, it is relatively straightforward to solve the *absolute orientation* problem relating camera coordinates to object coordinates. The solution is based on noting that the centroid of a rigid set of points is invariant to coordinate transformations. Let ${}^t\mathbf{P}_1, {}^t\mathbf{P}_2, \ldots {}^t\mathbf{P}_n$ and ${}^c\hat{\mathbf{P}}_1, {}^c\hat{\mathbf{P}}_2, \ldots, {}^c\hat{\mathbf{P}}_n$ denote n reference points in object coordinates and their corresponding

estimates in camera coordinates. Define $\overline{{}^t\mathbf{C}}$ and $\overline{{}^c\mathbf{C}}$ be the centroids these point sets, respectively, and define $\overline{{}^t\mathbf{P}_i} = {}^t\mathbf{P}_i - \overline{{}^t\mathbf{C}}$ and $\overline{{}^c\mathbf{P}_i} = {}^c\hat{\mathbf{P}}_i - \overline{{}^c\mathbf{C}}$. Then we have

$$ {}^c\mathbf{x}_t \circ ({}^t\mathbf{P}_i - \overline{{}^t\mathbf{C}}) - ({}^c\hat{\mathbf{P}}_i - \overline{{}^c\mathbf{C}}) = ({}^c\mathbf{R}_t {}^t\mathbf{P}_i + {}^c\mathbf{t}_t - {}^c\mathbf{R}_t \overline{{}^t\mathbf{C}} - {}^c\mathbf{t}_t) - ({}^c\hat{\mathbf{P}}_i - \overline{{}^c\mathbf{C}}) = {}^c\mathbf{R}_t \overline{{}^t\mathbf{P}_i} - \overline{{}^c\mathbf{P}_i}. $$

Note that the final expression depends only on ${}^c\mathbf{R}_t$. The corresponding least-squares problem can either be solved explicitly for ${}^c\mathbf{R}_t$ (see [56–58]), or solved incrementally using linearization. Given an estimate for ${}^c\mathbf{R}_t$, the computation of ${}^c\mathbf{t}_t$ is a linear least squares problem.

4.4 Discussion

The principle advantage of position-based control is that it is possible to describes tasks in terms of positioning in Cartesian coordinates. It's primary disadvantage is that it is often highly calibration dependent. The impact of calibration dependency often depends on the situation. In an environment where moderate positioning accuracy is required from firmly mounted cameras, extant calibration techniques probably provide a sufficiently accurate solution. However, if the cameras are moving and high accuracy is required, calibration sensitivity is an important issue.

Computation time for the relative orientation problem is often cited as a disadvantage of position-based methods. However recent results show that solutions can be computed in only a few milliseconds even using iteration [39] or Kalman filtering [37].

Endpoint closed-loop systems are demonstrably less sensitive to calibration. However, particularly in stereo systems, small rotational errors between the cameras can lead to reconstruction errors which do impact the positioning accuracy of the system. Thus, endpoint closed-loop systems will work well, for example, with a moving stereo head in which the cameras are fixed and rigid. However, it still may cause problems when both cameras are free to move relative to one another.

Feature-based approaches tend to be more appropriate to tasks where there is no prior model of the geometry of the task, for example in teleoperation applications [59]. Pose-based approaches inherently depend on an existing object model.

The pose estimation problems inherent in many position-based servoing problems requires solution to a potentially difficult correspondence problem. However, if the features are being tracked (see Section 6), then this problem need only be solved once at the beginning of the control process.

Many of these problems can be circumvented by sensing target pose directly using a 3D sensor. Active 3D sensors based on structured lighting are now compact and fast enough to use for visual servoing. If the sensor is small and mounted on the robot [60–62] the depth and orientation information can be used for position-based visual servoing.

5 Image-Based Control

As described in Section 3, in image-based visual servo control the error signal is defined directly in terms of image feature parameters (in contrast to position-based methods that define the error signal in the task space coordinates). Thus, we posit the following definition.

Definition 5.1 An *image-based visual servoing task* is represented by an image error function $\mathbf{e} : \mathcal{F} \rightarrow \Re^l$, where $l \leq k$ and k is the dimension of the image feature parameter space.

As described in Section 2.5, the system may use either a fixed camera or an eye-in-hand configuration. In either case, motion of the manipulator causes changes to the image observed by the vision system. Thus, the specification of an image-based visual servo task involves determining an appropriate error function $\mathbf{e}$, such that when the task is achieved, $\mathbf{e} = \mathbf{0}$. This can be done by directly using the projection equations (15), or by using a "teaching by showing" approach, in which the robot is moved to a goal position and the corresponding image is used to compute a vector of desired feature parameters, $\mathbf{f}_d$. If the task is defined with respect to a moving object, the error, $\mathbf{e}$, will be a function, not only of the pose of the end-effector, but also of the pose of the moving object.

Although the error, $\mathbf{e}$, is defined on the image parameter space, the manipulator control input is typically defined either in joint coordinates or in task space coordinates. Therefore, it is necessary to relate changes in the image feature parameters to changes in the position of the robot. The image Jacobian, introduced in Section 5.1, captures these relationships. We present an example image Jacobian in Section 5.2. In Section 5.3, we describe methods that can be used to "invert" the image Jacobian, to derive the robot velocity that will produce a desired change in the image. Finally, in Sections 5.4 and 5.5 we describe how controllers can be designed for image-based systems.

5.1 The Image Jacobian

In visual servo control applications, it is necessary to relate differential changes in the image feature parameters to differential changes in the position of the manipulator. The image Jacobian captures these relationships. Let $\mathbf{r}$ represent coordinates of the end-effector in some parameterization of the task space $\mathcal{T}$ and $\dot{\mathbf{r}}$ represent the corresponding end-effector velocity (note, $\dot{\mathbf{r}}$ is a velocity screw, as defined in Section 2.2). Let $\mathbf{f}$ represent a vector of image feature parameters and $\dot{\mathbf{f}}$ the corresponding vector of image feature parameter velocities. The image Jacobian, $\mathbf{J}_v$, is a linear transformation from the tangent space of $\mathcal{T}$ at $\mathbf{r}$ to the tangent space of $\mathcal{F}$ at $\mathbf{f}$. In particular,

$$\dot{\mathbf{f}} = \mathbf{J}_v \dot{\mathbf{r}} \tag{35}$$

where $\mathbf{J}_v \in \Re^{k \times m}$, and

$$J_v(\mathbf{r}) = \left[\frac{\partial \mathbf{F}}{\partial \mathbf{r}}\right] = \begin{bmatrix} \frac{\partial v_1(\mathbf{r})}{\partial r_1} & \cdots & \frac{\partial v_1(\mathbf{r})}{\partial r_m} \\ \vdots & & \vdots \\ \frac{\partial v_k(\mathbf{r})}{\partial r_1} & \cdots & \frac{\partial v_k(\mathbf{r})}{\partial r_m} \end{bmatrix}. \tag{36}$$

Recall that m is the dimension of the task space, $\mathcal{T}$. Thus, the number of columns in the image

Jacobian will vary depending on the task.

The image Jacobian was first introduced by Weiss *et al.* [19], who referred to it as the *feature sensitivity matrix.* It is also referred to as the *interaction matrix* [10] and the **B** matrix [14,15]. Other applications of the image Jacobian include [9,12,13,22].

The relationship given by (35) describes how image feature parameters change with respect to changing manipulator pose. In visual servoing we are interested in determining the manipulator velocity, $\dot{\mathbf{r}}$, required to achieve some desired value of $\dot{\mathbf{f}}$. This requires solving the system given by (35). We will discuss this problem in Section 5.3, but first we present an example image Jacobian.

5.2 An Example Image Jacobian

Suppose that the end-effector is moving with angular velocity $\Omega(t)$ and translational velocity $\mathbf{T}$ (as described in Section 2.2) both with respect to the camera frame in a fixed camera system. Let $\mathbf{p}$ be a point rigidly attached to the end-effector. The velocity of the point $\mathbf{p}$, expressed relative to the camera frame, is given by

$$^c\dot{\mathbf{p}} = \Omega \times {}^c\mathbf{p} + \mathbf{T} \tag{37}$$

To simplify notation, let $^c\mathbf{p} = [x, y, z]^T$. Substituting the perspective projection equations (15) into (9) – (10) we can write the derivatives of the coordinates of $\mathbf{p}$ in terms of the image feature parameters u, v as

$$\dot{x} = z\omega_y - \frac{vz}{f}\omega_z + T_x \tag{38}$$

$$\dot{y} = \frac{uz}{f}\omega_z - z\omega_x + T_y \tag{39}$$

$$\dot{z} = \frac{z}{f}(v\omega_x - u\omega_u) + T_z. \tag{40}$$

Now, let $\mathbf{F} = [u, v]^T$, as above and using the quotient rule,

$$\dot{u} = \lambda\frac{z\dot{x} - x\dot{z}}{z^2} \tag{41}$$

$$= \frac{\lambda}{z^2}\{z[z\omega_y - \frac{vz}{\lambda}\omega_z + T_x] - \frac{uz}{\lambda}[\frac{z}{\lambda}(v\omega_x - u\omega_y) + T_z]\} \tag{42}$$

$$= \frac{\lambda}{z}T_x - \frac{u}{z}T_z - \frac{uv}{\lambda}\omega_x + \frac{\lambda^2 + u^2}{\lambda}\omega_y - v\omega_z \tag{43}$$

Similarly

$$\dot{v} = \frac{\lambda}{z}T_y - \frac{v}{z}T_z + \frac{-\lambda^2 - v^2}{\lambda}\omega_x + \frac{uv}{\lambda}\omega_y + u\omega_z \tag{44}$$

Finally, we may rewrite these two equations in matrix form to obtain

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} \frac{\lambda}{z} & 0 & \frac{-u}{z} & \frac{-uv}{\lambda} & \frac{\lambda^2+u^2}{\lambda} & -v \\ 0 & \frac{\lambda}{z} & \frac{-v}{z} & \frac{-\lambda^2-v^2}{\lambda} & \frac{uv}{\lambda} & u \end{bmatrix} \begin{bmatrix} T_x \\ T_y \\ T_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{45}$$

which is an important result relating image-plane velocity of a point to the relative velocity of the point with respect to the camera. Alternative derivations for this example can be found in a number of references including [63,64].

It is straightforward to extend this result to the general case of using $k/2$ image points for the visual control by simply stacking the Jacobians for each pair of image point coordinates

$$\begin{bmatrix} \dot{u}_1 \\ \dot{v}_1 \\ \vdots \\ \dot{u}_{k/2} \\ \dot{v}_{k/2} \end{bmatrix} = \begin{bmatrix} \frac{\lambda}{z_1} & 0 & \frac{-u_1}{z_1} & \frac{-u_1v_1}{\lambda} & \frac{\lambda^2+u_1^2}{\lambda} & -v_1 \\ 0 & \frac{\lambda}{z_1} & \frac{-v_1}{z_1} & \frac{-\lambda^2-v_1^2}{\lambda} & \frac{u_1v_1}{\lambda} & u_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\lambda}{z_{k/2}} & 0 & \frac{-u_{k/2}}{z_{k/2}} & \frac{-u_{k/2}v_{k/2}}{\lambda} & \frac{\lambda^2+u_{k/2}^2}{\lambda} & -v_{k/2} \\ 0 & \frac{\lambda}{z_{k/2}} & \frac{-v_{k/2}}{z_{k/2}} & \frac{-\lambda^2-v_{k/2}^2}{\lambda} & \frac{u_{k/2}v_{k/2}}{\lambda} & u_{k/2} \end{bmatrix} \begin{bmatrix} T_x \\ T_y \\ T_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}. \tag{46}$$

Finally, note that the Jacobian matrices given in (45) and (46) are functions of the distance from the camera focal center to the point being imaged (i.e., they are functions of z_i). For a fixed camera system, when the target is the end-effector these z values can be computed using the forward kinematics of the robot and the camera calibration information. For an eye-in-hand system, determining z can be more difficult. This problem is discussed further in Section 7.1.

5.3 Using the Image Jacobian to Compute End-Effector Velocity

Visual servo control applications typically require the computation of $\dot{\mathbf{r}}$, given as input $\dot{\mathbf{f}}$. Methods for computing $\dot{\mathbf{f}}$ are discussed in Section 6. Here, we focus on the determination of $\dot{\mathbf{r}}$, assuming that $\dot{\mathbf{f}}$ is given. There are three cases that must be considered: $k = m$, $k < m$, and $k > m$. We now discuss each of these.

When $k = m$ and $\mathbf{J}_v$ is nonsingular, $\mathbf{J}_v^{-1}$ exists. Therefore, in this case, $\dot{\mathbf{r}} = \mathbf{J}_v^{-1}\dot{\mathbf{f}}$. Such an approach has been used by Feddema [18], who also describes an automated approach to image feature selection in order to minimize the condition of $\mathbf{J}_v$.

When $k \neq m$, $\mathbf{J}_v^{-1}$ does not exist. In this case, assuming that $\mathbf{J}_v$ is full rank (i.e., $\text{rank}(\mathbf{J}_v) = \min(k, m)$), we can compute a least squares solution, which, in general, is given by

$$\dot{\mathbf{r}} = \mathbf{J}_v^+ \dot{\mathbf{f}} + (\mathbf{I} - \mathbf{J}_v^+ \mathbf{J}_v)\mathbf{b} \tag{47}$$

where $\mathbf{J}_v^+$ is a suitable pseudoinverse for $\mathbf{J}_v$, and $\mathbf{b}$ is an arbitrary vector of the appropriate dimension. The least squares solution gives a value for $\dot{\mathbf{r}}$ that minimizes the norm $\|\dot{\mathbf{f}} - \mathbf{J}_v \dot{\mathbf{r}}\|$.

We first consider the case $k > m$, that is, there are more feature parameters than task degrees of freedom. By the implicit function theorem [65], if, in some neighborhood of $\mathbf{r}$, $m \leq k$ and $\text{rank}(\mathbf{J}_v) = m$ (i.e., $\mathbf{J}_v$ is full rank), we can express the coordinates $f_{m+1} \ldots f_k$ as smooth functions of $f_1 \ldots f_m$. From this, we deduce that there are $k-m$ redundant visual features. Typically, this will result in a set of inconsistent equations (since the k visual features will be obtained from a computer vision system, and therefore will likely be noisy). In this case, the appropriate pseudoinverse is given by

$$\mathbf{J}_v^+ = (\mathbf{J}_v^T \mathbf{J}_v)^{-1} \mathbf{J}_v^T. \tag{48}$$

Here, we have $(\mathbf{I} - \mathbf{J}_v^+ \mathbf{J}_v) = 0$ (the rank of the null space of $\mathbf{J}_v$ is 0, since the dimension of the column space of $\mathbf{J}_v$, m, equals $\text{rank}(\mathbf{J}_v)$). Therefore, the solution can be written more concisely as

$$\dot{\mathbf{r}} = \mathbf{J}_v^+ \dot{\mathbf{f}}. \tag{49}$$

When $k < m$, the system is underconstrained. In the visual servo application, this implies that the we are not observing enough features to uniquely determine the object motion $\dot{\mathbf{r}}$, i.e., there are certain components of the object motion that can not be observed. In this case, the appropriate pseudoinverse is given by

$$\mathbf{J}_v^+ = \mathbf{J}_v^T (\mathbf{J}_v \mathbf{J}_v^T)^{-1}. \tag{50}$$

In general, for $k < m$, $(\mathbf{I} - \mathbf{J}_v^+ \mathbf{J}_v) \neq 0$, and all vectors of the form $(\mathbf{I} - \mathbf{J}_v^+ \mathbf{J}_v)\mathbf{b}$ lie in the null space of $\mathbf{J}_v$, which implies that those components of the object velocity that are unobservable lie in the null space of $\mathbf{J}_v$. In this case, the solution is given by (47). For example, as shown in [64], the null space of the image Jacobian given in (45), is spanned by the four vectors

$$\begin{bmatrix} u \\ v \\ \lambda \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ u \\ v \\ \lambda \end{bmatrix} \quad \begin{bmatrix} uvz \\ -(u^2+\lambda^2)z \\ \lambda v z \\ -\lambda^2 \\ 0 \\ u\lambda \end{bmatrix} \quad \begin{bmatrix} \lambda(u^2+v^2+\lambda^2)z \\ 0 \\ -u(u^2+v^2+\lambda^2)z \\ uv\lambda \\ -(u^2+\lambda^2)z \\ u\lambda^2 \end{bmatrix}. \tag{51}$$

In some instances, there is a physical interpretation for the vectors that span the null space of the image Jacobian. For example, the vector $[u, v, \lambda, 0, 0, 0]^T$ reflects that the motion of a point along a projection ray cannot be observed. The vector $[0, 0, 0, u, v, \lambda]^T$ reflects the fact that rotation

of a point on a projection ray about that projection ray cannot be observed. Unfortunately, not all basis vectors for the null space have such an obvious physical interpretation. The null space of the image Jacobian plays a significant role in hybrid methods, in which some degrees of freedom are controlled using visual servo, while the remaining degrees of freedom are controlled using some other modality [12].

5.4 Resolved-Rate Methods

The earliest approaches to image-based visual servo control [9, 19] were based on resolved-rate motion control [28], which we will briefly describe here. Suppose that the goal of a particular task is to reach a desired image feature parameter vector, $\mathbf{f}_d$. If the control input is defined as in Section 4 to be an end-effector velocity, then we have $\mathbf{u} = \dot{\mathbf{r}}$, and assuming for the moment that the image Jacobian is square and nonsingular,

$$\mathbf{u} = \mathbf{J}_v^{-1}(\mathbf{r})(\dot{\mathbf{f}}). \tag{52}$$

If we define the error function as $\mathbf{e}(\mathbf{f}) = \mathbf{f}_d - \mathbf{f}$, a simple proportional control law is given by

$$\mathbf{u} = \mathbf{K}\mathbf{J}_v^{-1}(\mathbf{r})\mathbf{e}(\mathbf{f}), \tag{53}$$

where $\mathbf{K}$ is a constant gain matrix of the appropriate dimension. For the case of a non-square image Jacobian, the techniques described in Section 5.3 would be used to compute for $\mathbf{u}$. Similar results have been presented in [12, 13].

5.5 Example Servoing Tasks

In this section, we revisit the problems that were described in Section 4.1. Here, we describe image-based solutions for these problems.

Point to Point Positioning Consider the task of bringing some point $\mathbf{P}$ on the manipulator to a desired stationing point $\mathbf{S}$. If two cameras are viewing the scene, a necessary and sufficient condition for $\mathbf{P}$ and $\mathbf{S}$ to coincide in the workspace is that the projections of $\mathbf{P}$ and $\mathbf{S}$ coincide in each image.

If we let $[u^l, v^l]^T$ and $[u^r, v^r]^T$ be the image coordinates for the projection of $\mathbf{P}$ in the left and right images, respectively, then we may take $\mathbf{f} = [u^l, v^l, u^r, v^r]^T$. If we let $\mathcal{T} = \Re^3$, $\mathbf{F}$ is a mapping from $\mathcal{T}$ to $\mathbf{R}^4$.

Let the projection of $\mathbf{S}$ have coordinates $[u_s^l, v_s^l]$ and $[u_s^r, v_s^r]$ in the left and right images. We then define the desired feature vector to be $\mathbf{f}_d = [u_s^l, v_s^l, u_s^r, v_s^r]^T$, yielding

$$\mathbf{e}_{pp}(\mathbf{f}) = \mathbf{f} - \mathbf{f}_d. \tag{54}$$

The image Jacobian for this problem can be constructed by using (45) for each camera (note that a coordinate transformation must be used for either the left or right camera, to relate the end-effector velocity screw to a common reference frame).

Point to Line Positioning Consider again the task in which some point $\mathbf{P}$ on the manipulator end-effector is to be brought to the *line joining* two fixed points $\mathbf{S_1}$ and $\mathbf{S_2}$ in the world.

If two cameras are viewing the workspace, it can be shown that a necessary and sufficient condition for $\mathbf{P}$ to be colinear with the line joining $\mathbf{S_1}$ and $\mathbf{S_2}$ is that the projection of $\mathbf{P}$ be colinear with the projections of the points $\mathbf{S_1}$ and $\mathbf{S_2}$ in *both* images (for non-degenerate camera configurations). The proof proceeds as follows. The origin of the coordinate frame for the left camera, together with the projections of $\mathbf{S_1}$ and $\mathbf{S_2}$ onto the left image forms a plane. Likewise, the origin of the coordinate frame for the right camera, together with the projections of $\mathbf{S_1}$ and $\mathbf{S_2}$ onto the right image forms a plane. The intersection of these two planes is exactly the line joining $\mathbf{S_1}$ and $\mathbf{S_2}$ in the workspace. When $\mathbf{P}$ lies on this line, it must lie simultaneously in both of these planes, and therefore, must be colinear with the the projections of the points $\mathbf{S_1}$ and $\mathbf{S_2}$ in both images.

We now turn to conditions that determine when the projection of $\mathbf{P}$ is colinear with the the projections of the points $\mathbf{S_1}$ and $\mathbf{S_2}$. It is known that three vectors are coplanar if and only if their scalar triple product is zero. For the left image, let the projection of $\mathbf{S_1}$ have image coordinates $[u_1^l, v_1^l]$, the projection of $\mathbf{S_2}$ have image coordinates $[u_2^l, v_2^l]$, and the projection of $\mathbf{P}$ have image coordinates $[u^l, v^l]$. If the three vectors from the origin of the left camera to these image points are coplanar, then the three image points are colinear. Thus, we construct the scalar triple product

$$e_{pl}^l([u^l, v^l]^T) = \left(\begin{bmatrix} u_1^l \\ v_1^l \\ \lambda \end{bmatrix} \times \begin{bmatrix} u_2^l \\ v_2^l \\ \lambda \end{bmatrix} \right) \cdot \begin{bmatrix} u^l \\ v^l \\ \lambda \end{bmatrix}. \tag{55}$$

We may proceed in the same fashion to derive conditions for the right image

$$e_{pl}^r([u^r, v^r]^T) = \left(\begin{bmatrix} u_1^r \\ v_1^r \\ \lambda \end{bmatrix} \times \begin{bmatrix} u_2^r \\ v_2^r \\ \lambda \end{bmatrix} \right) \cdot \begin{bmatrix} u^r \\ v^r \\ \lambda \end{bmatrix}. \tag{56}$$

Finally, we construct the error function

$$\mathbf{e}(\mathbf{f}) = \begin{bmatrix} e_{pl}^l([u^l, v^l]^T) \\ e_{pl}^r([u^r, v^r]^T) \end{bmatrix} \tag{57}$$

where $\mathbf{f} = [u^l, v^l, u^r, v^r]^T$. Again, the image Jacobian for this problem can be constructed by using (45) for each camera (note that a coordinate transformation must be used for either the left or right camera, to relate the end-effector velocity screw to a common reference frame).

Given a second point on the end-effector, a four degree of freedom positioning operation can be defined by simply stacking the error terms. It is interesting to note that these solutions to the point-to-line problem perform with an accuracy that is independent of calibration, whereas the position-based versions do not [66].

5.6 Discussion

One of the chief advantages to image-based control over position-based control is that the positioning accuracy of the system is less sensitive camera calibration. This is particularly true for ECL image-based systems. For example, it is interesting to note that the ECL image-based solutions to the point-to-line positioning problem perform with an accuracy that is independent of calibration, whereas the position-based versions do not [66].

It is important to note, however, that most of the image-based control methods appearing in the literature still rely on an estimate of point position or target pose to parameterize the Jacobian. In practice, the unknown parameter for Jacobian calculation is distance from the camera. Some recent papers present adaptive approaches for estimating [14] this depth value, or develop feedback methods which do not use depth in the feedback formulation [67].

There are often computational advantages to image-based control, particularly in ECL configurations. For example, a position-based relative pose solution for an ECL single-camera system must perform two nonlinear least squares optimizations in order to compute the error function. The comparable image-based system must only compute a simple image error function, an inverse Jacobian solution, and possibly a single position or pose calculation to parameterize the Jacobian.

One disadvantage of image-based methods over position-based methods is the presence of singularities in the feature mapping function which reflect themselves as unstable points in the inverse Jacobian control law. These instabilities are often less prevalent in the equivalent position-based scheme. Returning again to the point-to-line example, the Jacobian calculation becomes singular when the two stationing points are coplanar with the optical centers of both cameras. In this configuration, rotations and translations of the setpoints in the plane are not observable. This singular configuration does not exist for the position-based solution.

In the above discussion we have referred to $\mathbf{f}_d$ as the desired feature parameter vector, and implied that it is a constant. If it is a constant then the robot will move to the desired pose with respect to the target. If the target is moving the system will endeavour to track the target and maintain relative pose, but the tracking performance will be a function of the system dynamics as discussed in Section 7.2.

Many tasks can be described in terms of the motion of image features, for instance aligning visual cues in the scene. Jang et al. [68] describe a generalized approach to servoing on image features, with trajectories specified in feature space – leading to trajectories (tasks) that are independent of target geometry. Skaar et al. [16] describes the example of a 1DOF robot catching a ball. By observing visual cues such as the ball, the arm's pivot point, and another point on the arm, the interception task can be specified, even if the relationship between camera and arm is not known a priori. Feddema [9] uses a feature space trajectory generator to interpolate feature parameter

values due to the low update rate of the vision system used.

6 Image Feature Extraction and Tracking

Irrespective of the control approach used, a vision system is required to extract the appropriate *task-relevant* features in order to perform the servoing task. Visual servoing pre-supposes the solution to a set of potentially difficult static and dynamic vision problems. To this end many reported implementations contrive the vision problem to be simple: *e.g.* painting objects white, using artificial targets, and so forth [9, 12, 54, 69]. Other authors use extremely task-specific clues: *e.g.* Allen [53] uses motion detection for locating a moving object to be grasped; welding systems commonly use special filters that isolate the image of the welding tip. A review of tracking approaches used by researchers in this field is given in [3].

In less structured situations, vision has typically relied on the extraction of sharp contrast changes, referred to as "corners" or "edges", to indicate the presence of object boundaries or surface markings in an image. Processing the entire image to extract these features necessitates the use of extremely high-speed hardware in order to work with a sequence of images at camera rate. However not all pixels in the image are of interest, and computation time can be greatly reduced if only a small region around each image feature is processed. Thus, a promising technique for making vision cheap and tractable is to use *window-based* tracking techniques [54, 70, 71]. Window-based methods have several advantages, among them; computational simplicity, little requirement for special hardware, and easy reconfiguration for different applications. We note, however, that often initialization of window or region-based systems still presupposes an automated or human-supplied solution to a potentially complex vision problem.

In keeping with the minimalist approach of this tutorial, we concentrate on describing the window-based approach to tracking of features in an image. A discussion of methods which use specialized hardware combined with temporal and geometric constraints can be found in [72]. The remainder of this section is organized as follows. Section 6.1 describes how window-based methods can be used to implement fast detection of edge segments, a common low-level primitive for vision applications. Section 6.2 describe an approach based on temporally correlating image regions over time. Section 6.3 describes some general issues related to the use of temporal and geometric constraints, and Section 6.4 briefly summarizes some of the issues surrounding the choice of a feature extraction method for tracking.

6.1 Feature based methods

The idea behind feature-based tracking methods is that at each stage of tracking a *pattern recognition* problem is solved. In this section, we illustrate how window-based processing techniques can be used to perform detection and tracking of isolated straight edge segments of fixed length. Edge segments are intrinsic to applications where man-made parts contain corners or other patterns formed from physical edges and vertices.

Images are comprised of pixels organized into a two-dimensional coordinate system. We adopt

the notation $I(\mathbf{x},t)$ to denote the pixel at location $\mathbf{x} = [u,v]^T$ in an image captured at time t. A window can be thought of as a two-dimensional array of pixels related to a larger image by an invertible mapping from window coordinates to image coordinates. We consider rigid transformations consisting of a translation vector $\mathbf{c} = [x,y]^T$ and a rotation θ. A pixel value at $\mathbf{x} = [u,v]^T$ in window coordinates is related to the larger image by

$$\mathcal{R}(\mathbf{x};\mathbf{c},\theta,t) = I(\mathbf{c} + R(\theta)\mathbf{x},t) \tag{58}$$

where R is a two dimensional rotation matrix. We adopt the convention that $\mathbf{x} = \mathbf{0}$ is the center of the window. In the sequel, the set $\mathcal{X}$ represents the set of all values of $\mathbf{x}$.

Window-based tracking algorithms typically operate in two stages. In the first stage, one or more windows are *acquired* using a nominal set of window parameters. The pixel values for all $\mathbf{x} \in \mathcal{X}$ are copied into a two-dimensional array that is subsequently treated as a rectangular image. Such acquisitions can be implemented extremely efficiently using line-drawing and region-fill algorithms commonly developed for graphics applications [73]. In the second stage, the windows are processed to locate features. Using feature measurements, a new set of window parameters are computed. These parameters may be modified using external geometric constraints or temporal prediction, and the cycle repeats.

We consider an edge segment to be characterized by three parameters in the image plane: the u and v coordinates of the center of the segment, and the orientation of the segment relative to the image plane coordinate system. These values correspond directly to the parameters of the acquisition window used for edge detection. Let us first assume we have correct prior values u^-, v^-, and θ^- for an edge segment. A window, $\mathcal{R}_0$, is extracted with these parameters would then have a vertical edge segment within it.

Isolated step edges can be localized by determining the location of the maximum of the first derivative of the signal [64,72,74]. However, since derivatives tend to increase the noise in an image, most edge detection methods combine spatial derivatives with a smoothing operation to suppress spurious maxima. Both derivatives and smoothing are linear operations that can be computed using convolution operators. Recall that the two-dimensional convolution of a window $\mathcal{R}(\cdot;\mathbf{c},\theta,t)$ by a function G is given by

$$(\mathcal{R} * G)(\mathbf{x}) = \frac{1}{|\mathcal{X}|}\sum_{\mathbf{s}\in\mathcal{X}} \mathcal{R}(\mathbf{x}-\mathbf{s};\mathbf{c},\theta,t)G(\mathbf{s}).$$

By the associativity of linear operations, the derivative of a smoothed signal is equivalent to the signal convolved with the derivative of the smoothing function. Hence, smoothing and differentiation can be combined into a single convolution template. Many edge detection methods based on smoothed derivatives have been proposed. An extremely popular convolution kernel is the derivative of a Gaussian (DOG) [75]. In one dimension, the DOG is defined as

$$g(x) = -x\exp(-x^2/\sigma^2)$$

where σ is a design parameter governing the amount of smoothing that takes place. Although the DOG has been demonstrated to be the optimal filter for detecting step edges [75], it takes floating

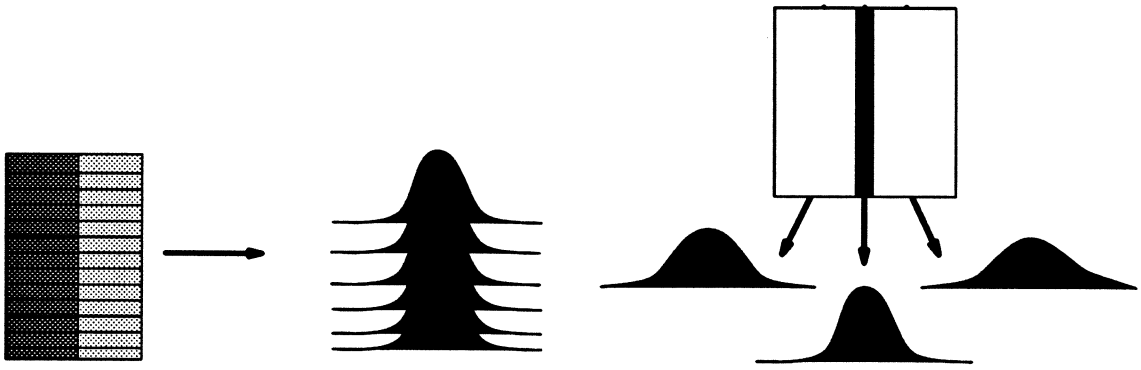

Figure 7: Window-based edge detection.

point arithmetic to compute accurately. Another edge detector which can be implemented without floating point arithmetic is the derivative of a triangle (DOT) kernal. In one dimension the DOT is defined as

$$g(x) = \text{signum}(x).$$

For a kernal three pixels wide, this is also known as the Prewitt operator [64]. Although the latter is not optimal from a signal processing point of view, convolution by the DOT can be implemented using only four additions per pixel. Thus, it is extremely fast to execute on simple hardware.

Convolutions are employed as follows. Let e any derivative-based scalar edge detection kernal arranged as a single row. Compute a new image $\mathcal{R}_1 = \mathcal{R}_0 * e$. $\mathcal{R}_1$ will have a response curve in each row which peaks at the location of the edge. Summing each column of $\mathcal{R}_1$ superimposes the peaks and yields a one-dimensional response curve. If the estimated orientation, θ^- was correct, the maximum of this response curve determines the offset of the edge in window coordinates. By interpolating the response curve, subpixel localization of the edge location can be achieved.

If the θ^- was incorrect, the response curves in $\mathcal{R}_1$ will deviate slightly from one another, and the final response curve will spread. Thus, the maximum value of the response curve is largest when the orientation of the edge in the original image matches the orientation of the detection window. Hence, by performing the detection operation on windows acquired at θ^- as well as two bracketing angles $\theta^- \pm \alpha$, quadratic interpolation on the maxima of the corresponding aggregate response curves can be used to estimate the orientation of the edge within the window.

As illustrated in Figure 7, computing the three oriented edge detectors is particularly simple if the range of angles is small. In this case, a single window is processed with the initial scalar convolution yielding $\mathcal{R}_1$. The response curves are computed by summing along the columns of $\mathcal{R}_1$, and in diagonal directions corresponding to angles of $\pm\alpha$. Thus, for the price of one window acquisition, one complete scalar convolution, and three column sums, the vertical offset δo and the orientation offset $\delta\theta$ can be computed. Once these two values are determined, the state variables of the acquisition window are updated as

$$\begin{aligned}\theta^+ &= \theta^- + \delta\theta \\ u^+ &= u^- - \delta o \sin(\theta^+) \\ v^+ &= v^- + \delta o \cos(\theta^+)\end{aligned}$$

On a Sun Sparc II, we have found that localizing a 20 pixel edge using a Prewitt-style mask 15 pixels wide searching ± 10 pixels and ± 15 degrees takes 1.5 ms on a Sun Sparc II workstation. At this rate, 22 edge segments can be tracked simultaneously at 30 Hz, the video frame rate used.

Clearly, this edge-detection scheme is susceptible to mistracking caused by background or foreground occluding edges. Large acquisition windows increase the range of motions that can be tracked, but reduce the tracking speed and increase the likelihood that a distracting edge will disrupt tracking. Likewise, large orientation brackets reduce the accuracy of the estimated orientation, and make it more susceptible to edges that are not closely oriented to the underlying edge.

There are several ways of increasing the robustness of edge tracking. One is to include some type of temporal component in the algorithm. For example, matching edges based on the sign or absolute value of the edge response increases its ability to reject incorrect edges. For more complex edge-based detection, collections of such oriented edge detectors can be combined to verify the location and position of the entire feature. Some general ideas in this direction are discussed in Section 6.3

6.2 Area Correlation-Based Methods

Feature-based methods do not make strong use of the actual appearance of the features they are tracking. If the desired feature is a specific pattern that changes little over time, then tracking can be based on correlating the appearance of the feature (in terms of its gray-values) in a series of images. SSD (Sum of Squared Differences) is a variation on correlation which tracks a region of an image by exploiting its *temporal consistency* — the observation that the appearance of small region in an image sequence changes very little.

Consider only windows that differ in the location of their center. We assume some reference window was acquired at time t at location $\mathbf{c}$. Some small time interval, τ, later a candidate window of the same size acquired at location $\mathbf{c} + \mathbf{d}$. The correspondence between these two images is measured by the sum of the squared differences of corresponding pixels:

$$O(\mathbf{d}) = \sum_{\mathbf{x} \in \mathcal{X}} (\mathcal{R}(\mathbf{x}; \mathbf{c}, t)) - \mathcal{R}(\mathbf{x}; \mathbf{c} + \mathbf{d}, t + \tau))^2 w(\mathbf{x}), \quad \tau > 0, \tag{59}$$

where $w(\cdot)$ is a weighting function over the image region.

The aim is to find the displacement, $\mathbf{d}$, that minimizes $O(\mathbf{d})$. Since images are inherently discrete, a natural solution is to select a finite range of values $\mathcal{D}$ and compute

$$\hat{\mathbf{d}} = \min_{\mathbf{d} \in \mathcal{D}} O(\mathbf{d}).$$

The advantage of a complete discrete search is that the true minimum over the search region is guaranteed to be found. However, the larger the area covered, the greater the computational burden. This burden can be reduced by performing the optimization starting at low resolution and proceeding to higher resolution, and by ordering the candidates in $\mathcal{D}$ from most to least likely and terminating the search once a candidate with an acceptably low SSD value is found [15]. Once the discrete minimum is found, the location can be refined to subpixel accuracy by interpolation of the SSD values about the minimum. Even with these improvements, [15] reports that a special signal processor is required to attain frame-rate performance.

It is also possible to solve (59) using continuous optimization methods [76–79]. The solution begins by expanding $\mathcal{R}(\mathbf{x};\mathbf{c},t)$ in a Taylor series about $(\mathbf{c},t)$ yielding

$$\mathcal{R}(\mathbf{x};\mathbf{c}+\mathbf{d},t+\tau) \approx \mathcal{R}(\mathbf{x};\mathbf{c},t) + \mathcal{R}_x(\mathbf{x},t)dx + \mathcal{R}_y(bx)dy + \mathcal{R}_t(\mathbf{x},t)\tau$$

where $\mathcal{R}_x$, $\mathcal{R}_y$ and $\mathcal{R}_t$ are the spatial and temporal derivatives of the image computed using convolution as follows:

$$\begin{aligned}
\mathcal{R}_x &= \mathcal{R} * \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} \\
\mathcal{R}_y &= \mathcal{R} * \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} \\
\mathcal{R}_t &= (\mathcal{R}(\cdot;\mathbf{c},t+\tau) - \mathcal{R}_s(\cdot;\mathbf{c},t)) * \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}
\end{aligned}$$

Substituting into (59) yields

$$O(\mathbf{d}) \approx \sum_{\mathbf{x}\in\mathcal{X}} \left(\mathcal{R}_x(\mathbf{x};\mathbf{c},t)dx + \mathcal{R}_y(\mathbf{x};\mathbf{c},t)dy + \mathcal{R}_t(\mathbf{x};\mathbf{c},t)\tau\right)^2 w(x) \tag{60}$$

Define

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} \mathcal{R}_x(\mathbf{x}) \\ \mathcal{R}_y(\mathbf{x}) \end{bmatrix}$$

Expression (60) can now be written more concisely as

$$O(\mathbf{d}) \approx \sum_{\mathbf{x}\in\mathcal{X}} \left(\mathbf{g}(\mathbf{x})\cdot\mathbf{d} + \mathcal{R}_t(\mathbf{x})\tau\right)^2 w(x). \tag{61}$$

Notice O is now a quadratic function of $\mathbf{d}$. Computing the derivatives of O with respect to the components of $\mathbf{d}$, setting the result equal to zero, and rearranging yields a linear system of equations:

$$\left[\sum_{\mathbf{x}\in\mathcal{X}} (\mathbf{g}(\mathbf{x})\mathbf{g}(\mathbf{x})^T w(x))\right] \mathbf{d} = \sum_{\mathbf{x}\in\mathcal{X}} \mathcal{R}_t(\mathbf{x})\mathbf{g}(\mathbf{x})w(x) \tag{62}$$

Solving for $\mathbf{d}$ yields an estimate, $\hat{\mathbf{d}}$ of the offset that would cause the two windows to have maximum correlation. We then compute $\mathbf{c}^+ = \mathbf{c}^- + \hat{\mathbf{d}}$ yielding the updated window location for the next

tracking cycle. This is effectively a proportional control algorithm for the "servoing" the location of an acquisition to maintain the best match with the reference window over time.

In practice this method will only work for small motions (it is mathematically correct only for a fraction of a pixel). This problem can be alleviated by first performing the optimization at low levels of resolution, and using the result as a seed for computing the offset at higher levels of resolution. For example, reducing the resolution by a factor of two by summing groups of four neighboring pixels doubles the maximum displacement between two images. It also speeds up the computations since fewer operations are needed to compute $\hat{\mathbf{d}}$ for the smaller low-resolution image.

Continuous optimization has two principle advantages over discrete optimization. First, a single updating cycle is usually faster to compute. For example, (62) can be computed and solved in less than 5 ms on a Sparc II computer [79]. Second, it is easy to incorporate other window parameters such as rotation and scaling into the system without greatly increasing the computation time [78,79]. It is also easy to show that including parameters for contrast and brightness in (60) makes SSD tracking equivalent to finding the maximum correlation between the two image regions [76]. Thus, SSD methods can be used to perform template matching as well as tracking of image regions.

6.3 Filtering and Feedforward

Window-based tracking implicitly assumes that the interframe motions of the tracked feature do not exceed the size of search window, or, in the case of SSD tracking, a few pixels from the expected location of the image region. In the simplest case, the previous location of the image feature can be used as a predictor of its current location. Unfortunately, as feature velocity increases the search window must be enlarged which adversely affects computation time.

The robustness and speed of tracking can be significantly increased with knowledge about the dynamics of the observed features, which may be due to motion of the camera or target. For example, given knowledge of the image feature location $\mathbf{x}_t$ at time t, Jacobian $\mathbf{J}_v$, the end-effector velocity $\mathbf{u}_t$, and the interframe time τ, the expected location of the search windows can be computed by the prediction

$$\mathbf{f}_{t+\tau} = \mathbf{f}_t + \tau J_v \mathbf{u}_t.$$

Likewise, if the dynamics of a moving object are known, then it is possible to use this to enhance prediction. For example, Rizzi [54] describes the use of a Newtonian flight dynamics model to make it possible to track a ping-pong ball during flight. Predictors based on $\alpha - \beta$ tracking filters and Kalman filters have also been used [37,53,72].

Multiresolution techniques can be used provide further performance improvements, particularly when a dynamic model is not available and large search windows must be used.

6.4 Discussion

Prior to executing or planning visually controlled motions, a specific set of visual features must be chosen. Discussion of the issues related to feature selection for visual servo control applications can be found in [19] [18]. The "right" image feature tracking method to use is extremely application

dependent. For example, if the goal is to track a single special pattern or surface marking that is approximately planar and moving at slow to moderate speeds, then SSD tracking is appropriate. It does not require special image structure (*e.g.* straight lines), it can accommodate a large set of image distortions, and for small motions can be implemented to run at frame rates.

In comparison to the edge detection methods described above, SSD tracking is extremely sensitive to background changes or occlusions. Thus, if a task requires tracking several occluding contours of an object with a changing background, edge-based methods are clearly faster and more robust.

In many realistic cases, neither of these approaches by themselves yields the robustness and performance desired. For example, tracking occluding edges in an extremely cluttered environment is sure to distract edge tracking as "better" edges invade the search window, while the changing background would ruin the SSD match for the region. Such situations call for the use of more global task constraints (*e.g.* the geometry of several edges), more global tracking (*e.g.* extended contours or snakes [80]), or improved or specialized detection methods.

To illustrate these tradeoffs, suppose a visual servoing task relies on tracking the image of a circular opening over time. In general, this hole will project to an ellipse in the camera. There are several window-based algorithms that could be used to detect this ellipse and recover its parameters.

1. If the contrast between the hole and the outside is high, then binary thresholding followed by a calculation of the first and second central moments can be used to localize the feature [54].

2. If the ambient illumination changes greatly over time, but the brightness of the hole and the brightness of the surrounding region are roughly constant, a circular template could be localized using SSD methods augmented with brightness and contrast parameters. Note that (59) must also include parameters for scaling and aspect ratio.

3. The hole could also be selected in an initial image, and subsequently located using SSD correlation. Note that this calculation does not compute the center of the hole, only its correlation. Although useful for servoing a camera to maintain the hole within the field of view, this approach is probably not useful for actual manipulation tasks.

4. If the contrast and background are changing, the hole could be tracked by performing edge detection and fitting an ellipse to the edge locations. In particular, short edge segments could be located using the techniques described above. Once the segments have been fit to an ellipse, the orientations and location of the segments would be adjusted for the subsequent tracking cycle.

During task execution, other problems arise. The two most common problems are occlusion of features and and visual singularities. Solutions to the former include intelligent observers that note the disappearance of features and continue to predict their locations based on dynamics and/or feedforward information [54], or redundant feature specifications that can perform even with some loss of information. Solution to the latter require some combination of intelligent path planning and/or intelligent acquisition and focus-of-attention to maintain the controllability of the system.

It is probably safe to say that image processing presents the greatest challenge to general-purpose hand-eye coordination.

7 Related Issues

In this section, we briefly discuss a number of related issues that were not addressed in the tutorial.

7.1 Image-Based versus Position-Based Control

The taxonomy of visual servo introduced in Section 1 has four major architectural classes. Most systems that have been reported fall into the dynamic position- or image-based look-and-move structure. That is, they employ axis-level feedback, generally of position, for reasons outlined earlier. No reports of an implementation of the position-based direct visual servo structure are known to the authors. Weiss's proposed image-based direct visual-servoing structure does away entirely with axis sensors — dynamics and kinematics are controlled adaptively based on visual feature data. This concept has a certain appeal but in practice is overly complex to implement and appears to lack robustness (see, e.g., [81] for an analysis of the effects of various image distortions on such control schemes). The concepts have only ever been demonstrated in simulation for up to 3-DOF and then with simplistic models of axis dynamics which ignore 'real world' effects such as Coulomb friction and stiction. Weiss showed that even when these simplifying assumptions were made, sample intervals of 3 ms were required. This would necessitate significant advances in sensor and processing technology, and the usefulness of controlling manipulator kinematics and dynamics this way must be open to question.

Many systems based on image-based and position-based architectures have been demonstrated, and the computational costs of the two approaches are comparable and readily achieved. The often cited advantage of the image-based approach, reduced computational burden, is doubtful in practice. Many reports are based on using a constant image Jacobian, which is computationally efficient, but valid only over a small region of the task space. The general problem of Jacobian update remains, and in particular there is the difficulty that many image Jacobians are a function of target depth, z. This necessitates a partial pose estimation which is the basis of the position-based approach. The cited computational disadvantages of the position-based approach have been ameliorated by recent research — photogrammetric solutions can now be computed in a few milliseconds, even using iteration.

7.2 Dynamic Issues in Closed-loop Systems

A visual servo system is a closed-loop discrete-time dynamical system. The sample rate is the rate at which images can be processed and is ultimately limited by the frame rate of the camera, though many reported systems operate at a sub-multiple of the camera frame rate due to limited computational ability. Negative feedback is applied to a *plant* which generally includes a significant time delay. The sources of this delay include; charge integration time within the camera, serial

pixel transport from the camera to the vision system, and computation time for feature parameter extraction. In addition most reported visual servo system employ a relatively low bandwidth communications link between the vision system and the robot controller, which introduces further latency. Some robot controllers operate with a sample interval which is not related to the sample rate of the vision system, and this introduces still further delay. A good example of this is the common Unimate Puma robot whose position loops operate at a sample interval of 14 or 28 ms while vision systems operate at sample intervals of 33 or 40 ms for RS 170 or CCIR video respectively [27].

It is well known that a feedback system including delay will become unstable as the loop gain is increased. Many visual closed-loop systems are tuned empirically, increasing the loop gain until overshoot or oscillation becomes intolerable. While such closed-loop systems will generally converge to the desired pose with a zero error, the same is not true when tracking a moving target. The tracking performance is a function of the closed-loop dynamics, and for simple proportional controllers will exhibit a very significant time lag or phase delay. If the target motion is constant then prediction (based upon some assumption of target motion) can be used to compensate for the latency, but combined with a low sample rate this results in poor disturbance rejection and long reaction time to target 'maneuvers'. Predictors based on autoregressive models, Kalman filters, $\alpha - \beta$ and $\alpha - \beta - \gamma$ tracking filters have been demonstrated for visual servoing. In order for a visual-servo system to provide good tracking performance for moving targets considerable attention must be paid to modelling the dynamics of the robot and vision system and designing an appropriate control system. Other issues for consideration include whether or not the vision system should 'close the loop' around robot axes which are position, velocity or torque controlled. A detailed discussion of these dynamic issues in visual servo systems is given by Corke [27, 82].

7.3 Mobile robots

The discussion above has assumed that the moving camera is mounted on an arm type robot manipulator. For mobile robots the pose of the robot is generally poorly known and can be estimated from the relative pose of known fixed objects or landmarks. Most of the techniques described above are directly applicable to the mobile robot case. Visual servoing can be used for navigation with respect to landmarks or obstacle and to control docking (see, e.g., [83]).

7.4 A Light-Weight Tracking and Servoing Environment

The design of many task-specific visual tracking and vision-based feedback systems used in visual servoing places a strong emphasis on system modularity and reconfigurability. This has motivated the development of a modular, software-based visual tracking system for experimental vision-based robotic applications [84]. The system design emphasizes flexibility and efficiency on standard scientific workstations and PC's. The system is intended to be a portable, inexpensive tool for rapid prototyping and experimentation for teaching and research.

The system is written as a set of classes in C++. The use of object-oriented methods hides the details of how specific methods are implemented, and structures applications through a pre-specified set of generic interfaces. It also enhances the portability of the system by supporting

device abstraction. The current system runs on several framegrabbers including models avialable from most common manufacturers. More information on the system and direction for retrieving it can be found at `http://www.cs.yale.edu/HTML/YALE/CS/AI/VisionRobotics/YaleAI.html`.

The same design philosophy is currently being used to develop a complementary hand-eye coordination toolkit which is to be avilable in the near future.

7.5 Current Research Problems

There are many open problems in visual servo control, too numerous to describe here. These include control issues, such as adaptive visual servo control [14,85], hybrid control (e.g., hybrid vision/position control [12], or hybrid force/vision control), and multi-rate system theory [86]; issues related to automatic planning of visually controlled robot motions [87,88]; applications in mobile robotics, including nonholonmoic systems [83]; and, feature selection [18,78]. Many of these are describe in the proceedings of a recent workshop on visual servo control [89].

7.6 The future

The future for applications of visual servoing should be bright. Camera's are relatively inexpensive devices and the cost of image processing systems continues to fall.

Most visual servo systems make use of cameras that conform to broadcast television standards, and this is now a significant limiting factor toward achieving high-performance visual servoing. Those standards were developed over 50 years ago with very specific design aims and to their credit still serve. Advances in digital image processing technology over the last decade have been rapid and vision systems are now capable of processing far more pixels per second than a camera can provide. Breaking this bottleneck would allow use of higher resolution images, higher frame rates or both. The current push for HDTV will have useful spinoffs for higher resolution visual servoing. New standards (such as promoted by the AIA) for digital output cameras are spurring the development of new cameras that are not tied to the frame rates and interlacing of the old broadcast standards.

Visual servoing also requires high-speed image processing and feature extraction. The increased performance and falling cost of computer vision systems, and computing systems in general, is a consequence of Moore's Law. This law, originally postulated in 1964 predicts that computer circuit density will double every year, and 30 years later still holds true.

Robust scene interpretation is perhaps the greatest, current, limitation to the wider application of visual servoing. Considerable progress is required if visual-servo systems are to move out of environments lined with black velvet.