Visual Servo Control

Seth Hutchinson
Georgia Institute of Technology

With contributions from

- Nicholas Gans -- University of Texas at Dallas
- Peter Corke -- Queensland University of Technology, Australia
- Sourabh Battacharya – Iowa State University

*includes material from the articles “Visual Servo Control, Part I: Basic Approaches,” and “Visual Servo Control, Part II: Advanced Approaches,” in the *IEEE Robotics and Automation Society Magazine*, by François Chaumette and Seth Hutchinson.

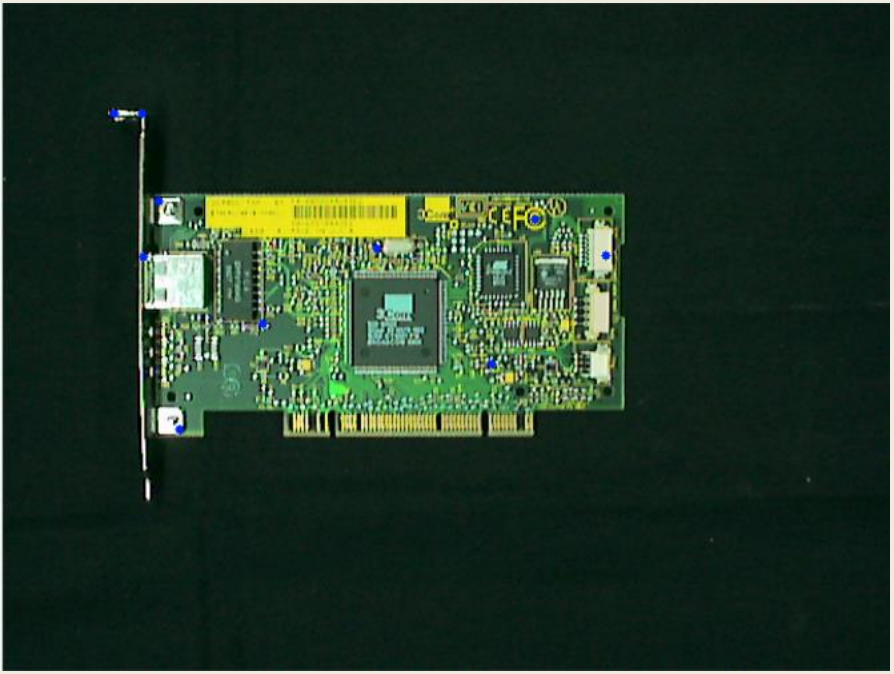
Visual Servo: The Basic Problem

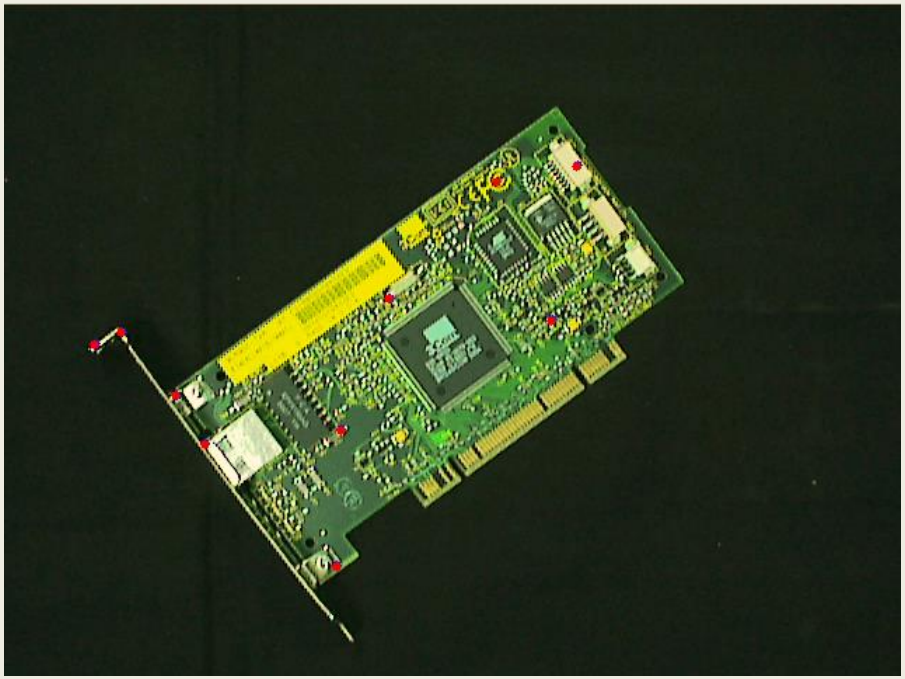
- A camera views the scene from an initial pose, yielding the *current image*.
- The *desired image* corresponds to the scene as viewed from the desired camera pose.
- Determine a camera motion to move from initial to desired camera pose, using the time-varying image as input.

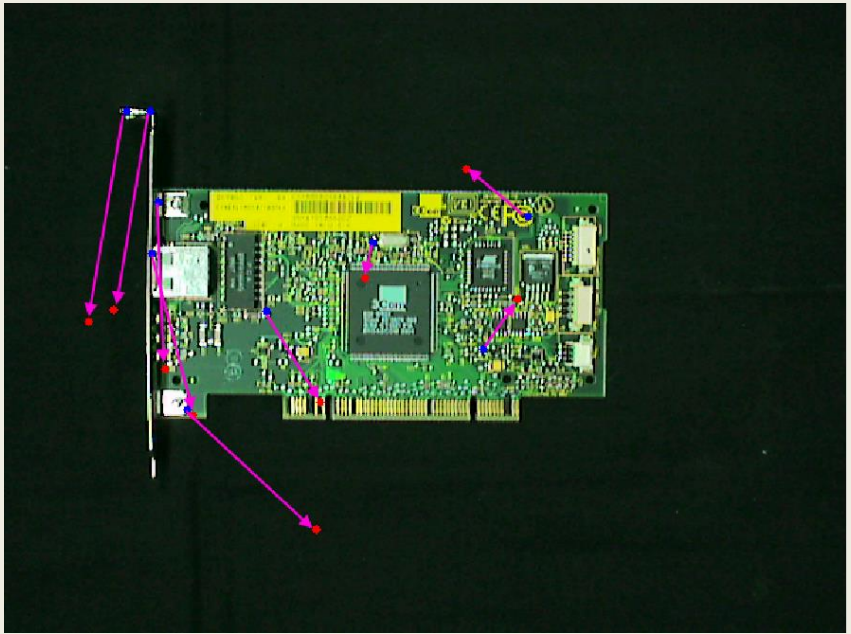
There are many variations on the problem:

- Eye-in-hand vs. fixed camera
- Which image features to use
- How to specify desired images for specified tasks
- Etc…

An Example







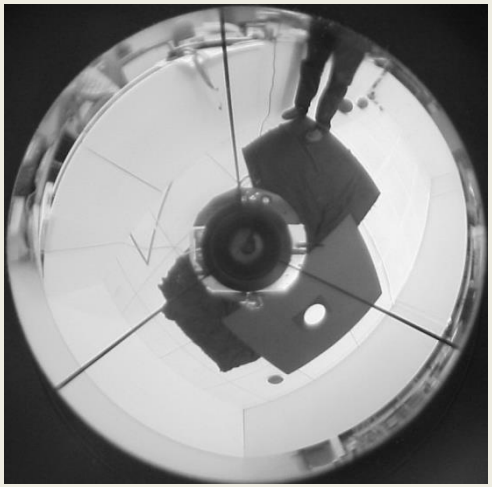
In this example, coordinates of image points are the features

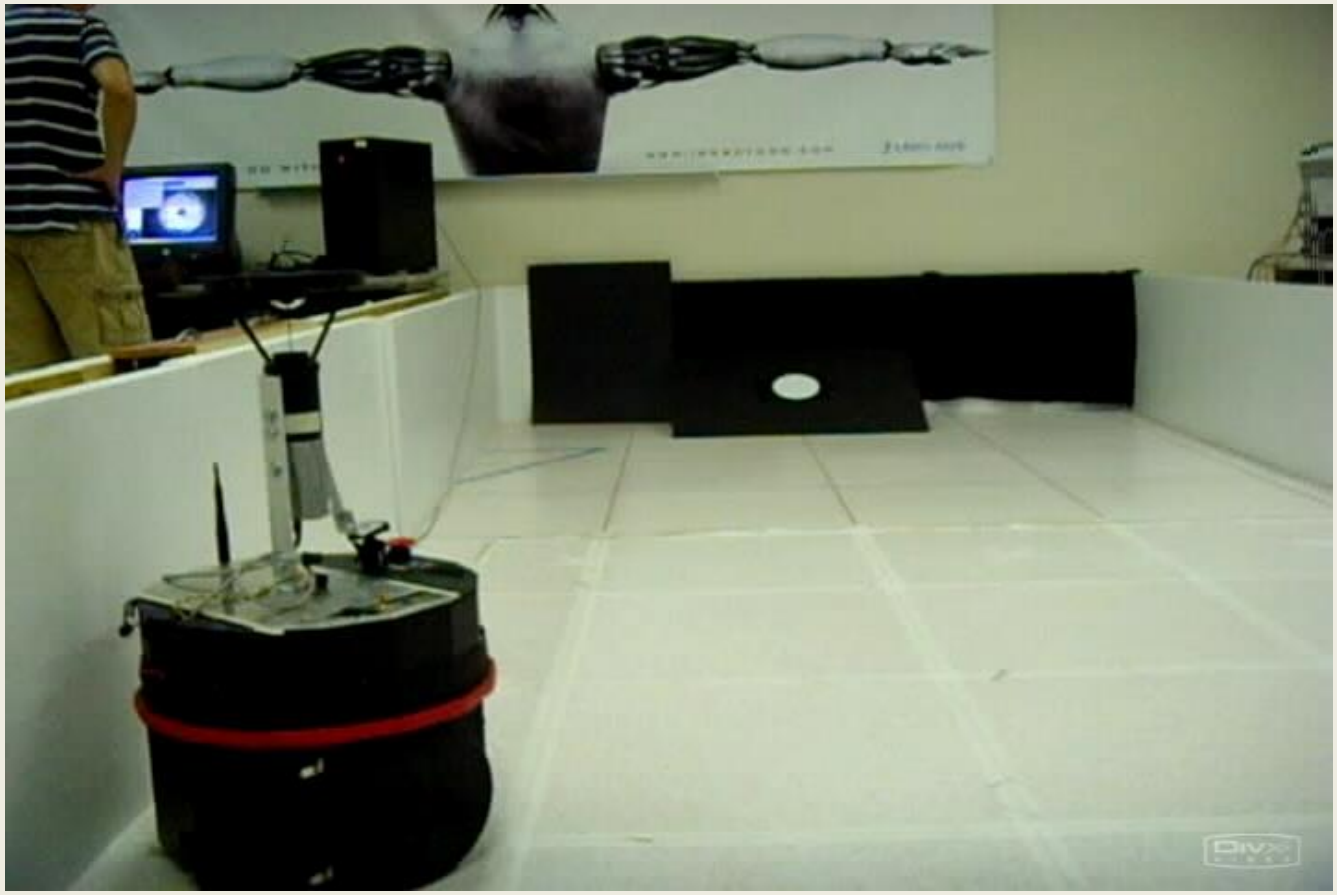
Blue points are current features

Red points are desired features

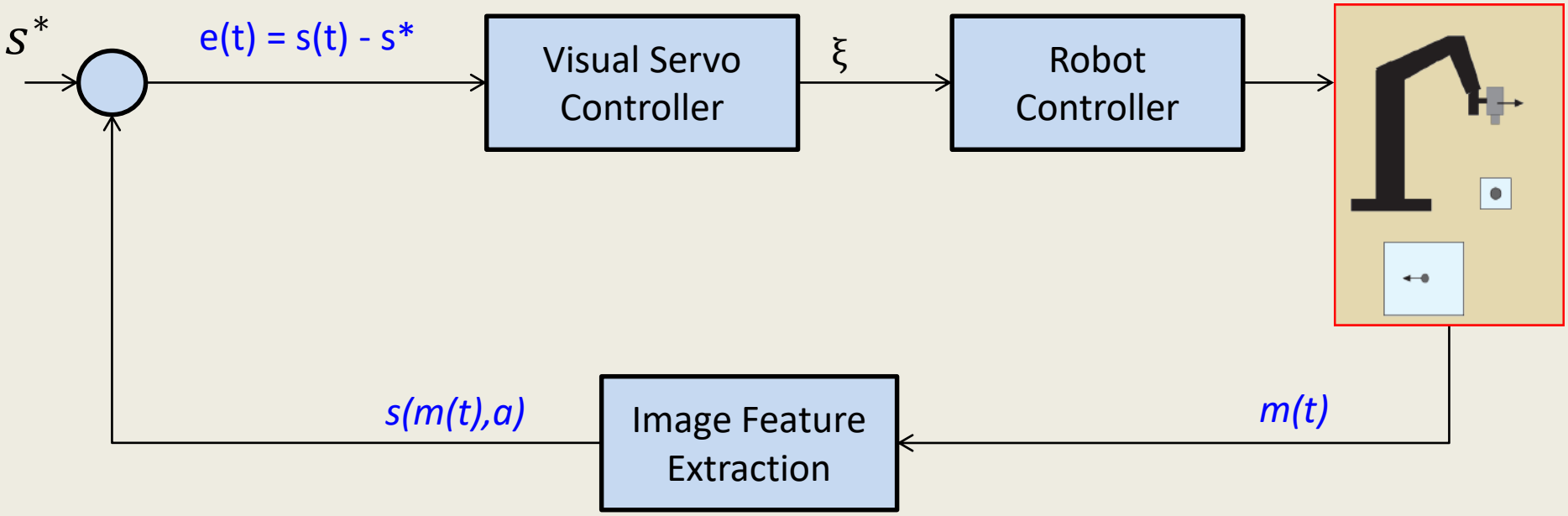
Error vectors are shown in pink

Coarsely Calibrated Visual Servoing of a Nonholonomic mobile robot Using a Central Catadioptric Vision System



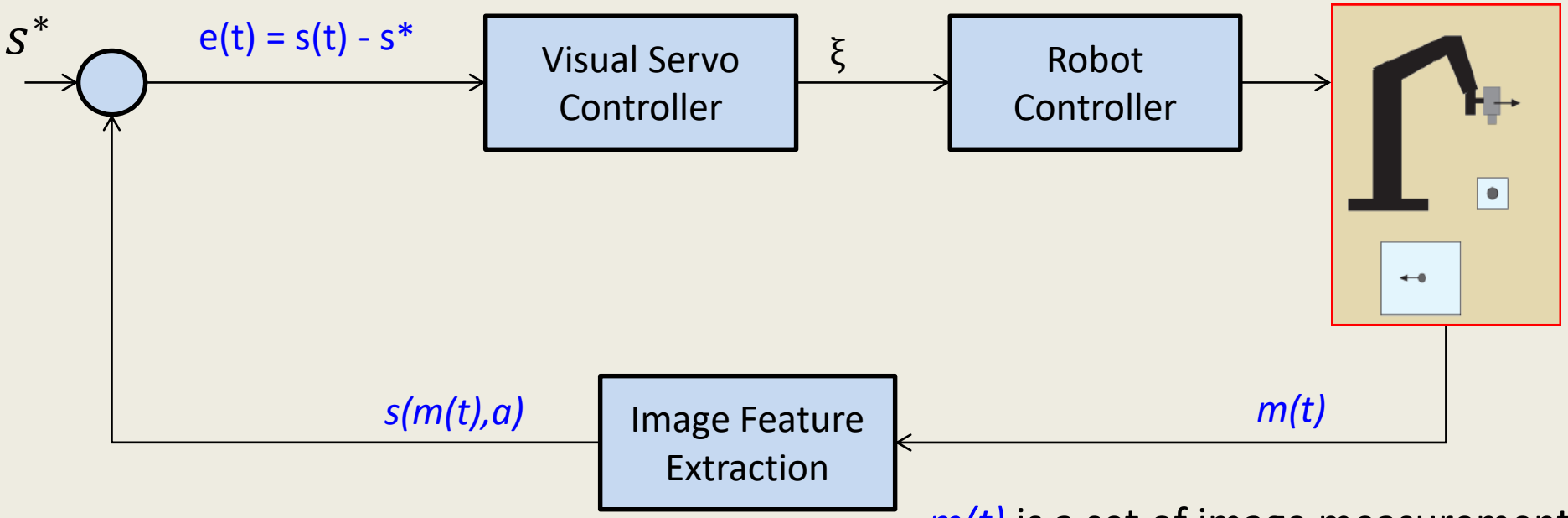


Romeo Tatsambon, and Andrea Cherubini --- IRISA, Rennes
Han Ul Yoon --- University of Illinois



Basic architecture for Visual Servo Control

At this level of abstraction, it's remarkably similar to the architecture for any garden variety feedback control system.



s(m(t),a) is a vector of k visual features, computed from the image.

a is a set of parameters that represent additional knowledge about the system (e.g., coarse camera intrinsic parameters or 3D model of objects).

s* contains the desired values of the features.

m(t) is a set of image measurements (e.g., the image coordinates of interest points, or the parameters of a set of image segments).

Error defined by e(t) = s(t) - s*

Velocity to camera = ξ

Visual Servo Control --- The Basic Idea

The aim of vision-based control schemes is to minimize an error *e(t)* which is typically defined by

$$e(t) = s(m(t), a) - s^*$$

- *m(t)* is a set of image measurements (e.g., the image coordinates of interest points, or the parameters of a set of image segments).
- *s(m(t),a)* is a vector of k visual features, computed from the image.
- *a* is a set of parameters that represent additional knowledge about the system (e.g., camera intrinsic parameters or 3D object model).
- s* contains the desired values of the features.

Typically, one merely writes: e(t) = s(t) - s*

Some Basic Assumptions

There are numerous considerations when designing a visual servo system. For now, we will consider only systems that satisfy the following basic assumptions:

- Eye-in-hand systems — the camera is mounted on the end effector of a robot and treated as a free-flying object with configuration space Q = SE(3).
- Static (i.e., motionless) targets.
- Purely kinematic systems — we do not consider the dynamics of camera motion, but assume that the camera can execute accurately the applied velocity control.
- Perspective projection — the imaging geometry can be modeled as a pinhole camera.

Some or all of these may be relaxed as we progress to more advanced topics.

Designing the Control Law --- The Basic Idea

Given s, control design can be quite simple.

A typical approach is to design a velocity controller, which requires the relationship between the time variation of *s* and the camera velocity.

- Let the spatial velocity of the camera be denoted by $\xi = (v, \omega)$
  - v is the instantaneous linear velocity of the origin of the camera frame
  - ω is the instantaneous angular velocity of the camera frame

- The relationship between $\dot{s}$ and ξ is given by $\dot{s} = L\xi$

  $L \in \mathbb{R}^{6\,x\,k}$ is named the *interaction matrix* [Espiau, et al. 1992], or the *image Jacobian* [Hutchinson, Hager & Corke, 1996].

$\Longrightarrow$ *The key to visual servo --- choosing s and the control law.*

Designing the Control Law (cont)

Let's derive the relationship between $\dot{e}$ and ξ, i.e., how does the error evolve as a function of the camera body velocity?

- Using the previous equations

$$e(t) = s(t) - s^* \quad \text{and} \quad \dot{s} = L\,\xi$$

we can easily obtain the relationship between the camera velocity and the rate of change of the error $\dot{e}$

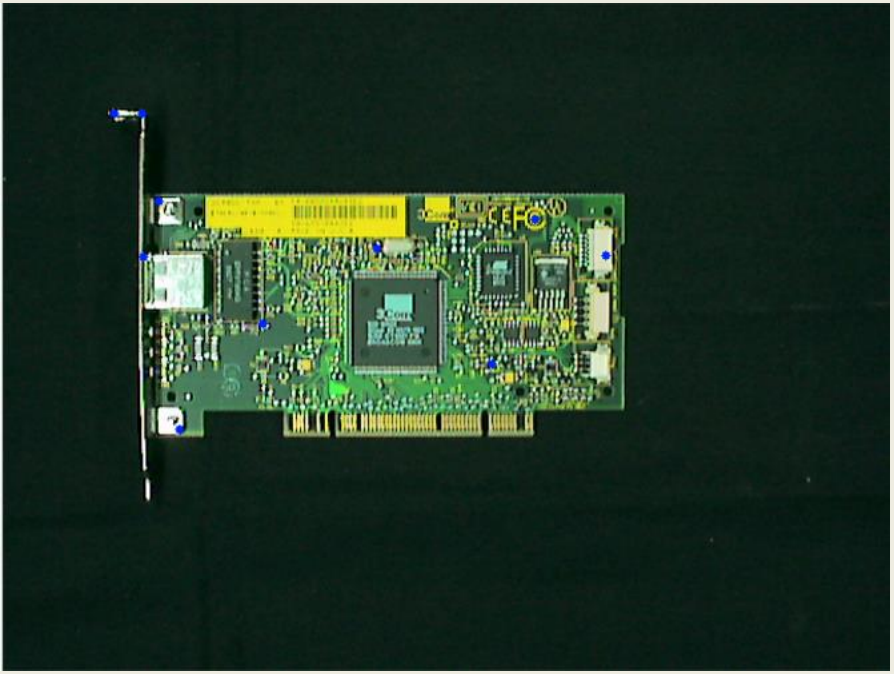
$$\dot{e}(t) = \dot{s}(t) = L\,\xi$$

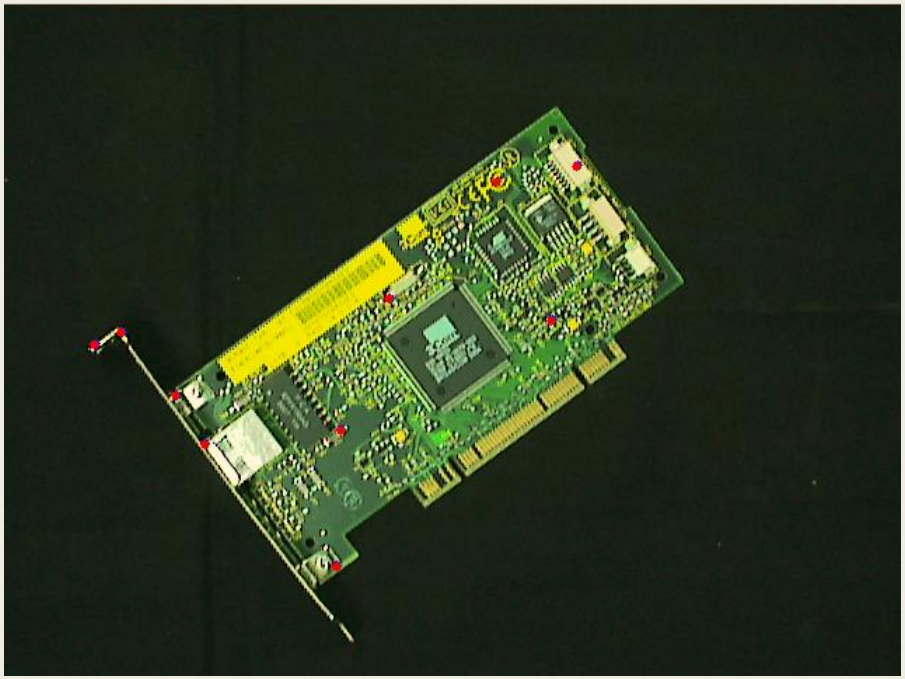
assuming that s^* is constant.

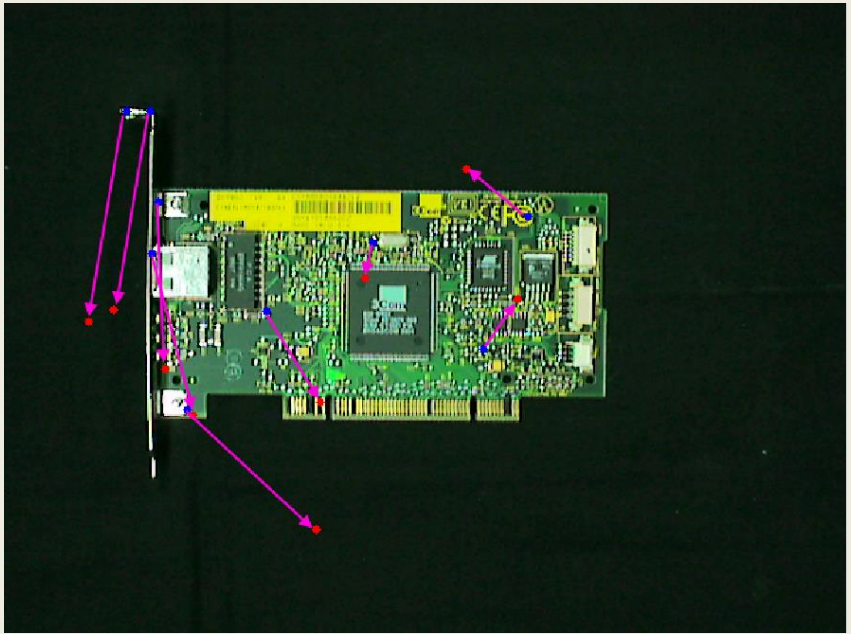
- The relationship between ξ and $\dot{s}$ is the same as between ξ and $\dot{e}$.

Now our problem is merely to find the control input $\xi = u(t)$ that gives the desired error performance.

An Example







In this example, coordinates of image points are the features

Blue points are current features

Red points are desired features

Error vectors are shown in pink

Designing the Control Law (cont)

- In many cases, we would like to achieve an exponential decoupled decrease of the error, $e(t) = \mathrm{e}(\mathrm{t}_0)\exp(-\lambda t)$

- This is achieved if the error obeys the ordinary differential equation

$$\dot{e}(t) = -\lambda e$$

- Combining $\dot{e}(t) = -\lambda e$ and $\dot{e}(t) = L\,\xi$ we obtain

$$L\xi = -\lambda\, e$$

- If we assume velocity control, i.e., $u(t) = \xi$, we simply solve the above to obtain

$$u(t) = \xi = -\lambda\, L^{+} e$$

where $L^{+} \in \mathbb{R}^{k\,x\,6}$ is chosen as the Moore-Penrose pseudo-inverse of L

$$L^{+} = (L^{T} L)^{-1} L^{T}$$

Practical Issues

In practice, it is impossible to know exactly the value of L or of L^+, since these depend on measured data.

The actual value of L is thus an approximation, and the actual control law is given as

$$\xi = -\lambda \widehat{L^+} e$$

There are several choices for $\widehat{L^+}$:

- $\widehat{L^+} = \hat{L}^+$: Compute an estimate $\hat{L}$, use the pseudo-inverse of the estimate
- Directly estimate $\widehat{L^+}$
- Let $\widehat{L^+}$ be approximated by a constant matrix (e.g., L^+ for the goal camera configuration)

Context – Visual Servo in the Bigger Picture

- Learning, planning, perception and action are often tightly coupled activities.
- Visual servo control is the coupling of perception and action

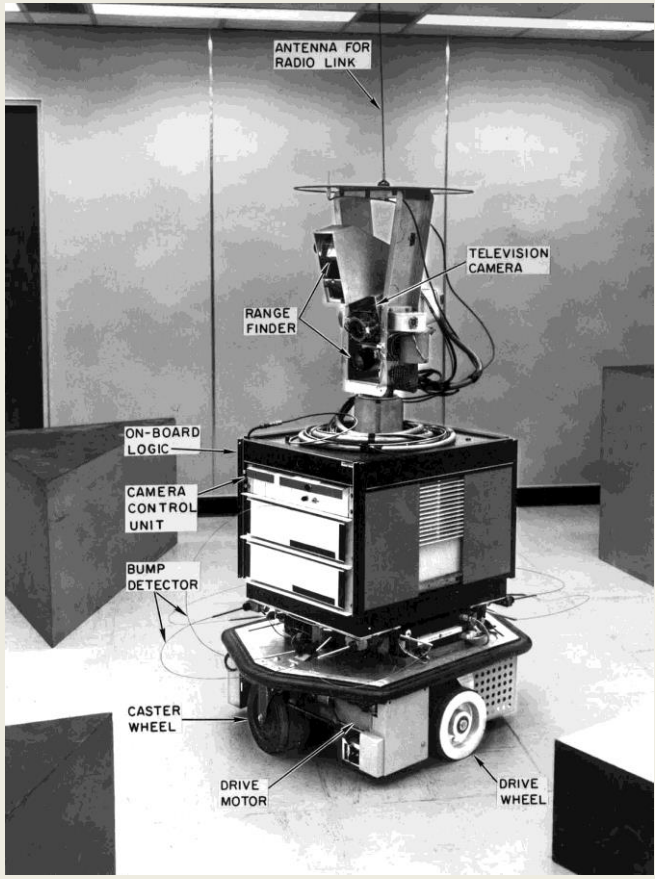
  — hand-eye coordination.
- Basic visual servo controllers can serve as primitives for planning algorithms.
- Switching between control laws is equivalent to executing a plan.
- There are a number of analogies between human hand-eye coordination and visual servo control.

A rigorous understanding of the performance of visual servo control systems provides a foundation for sensor-based robotics.

Visual Servo Control --- Some History

Visual servo control is merely the use of computer vision data to control motion of a robot

In some sense, Shakey [SRI, 1966-1972 or so] was an example of visual servo system, but with a very, very slow servo rate



The first real-time visual servo systems were reported in

- Agin, 1979
- Weiss et al., 1984, 1987
- Feddema et al., 1989

In each of these, simple image features (e.g., centroids of binary objects) were used, primarily due to limitations in computation power.

Overview

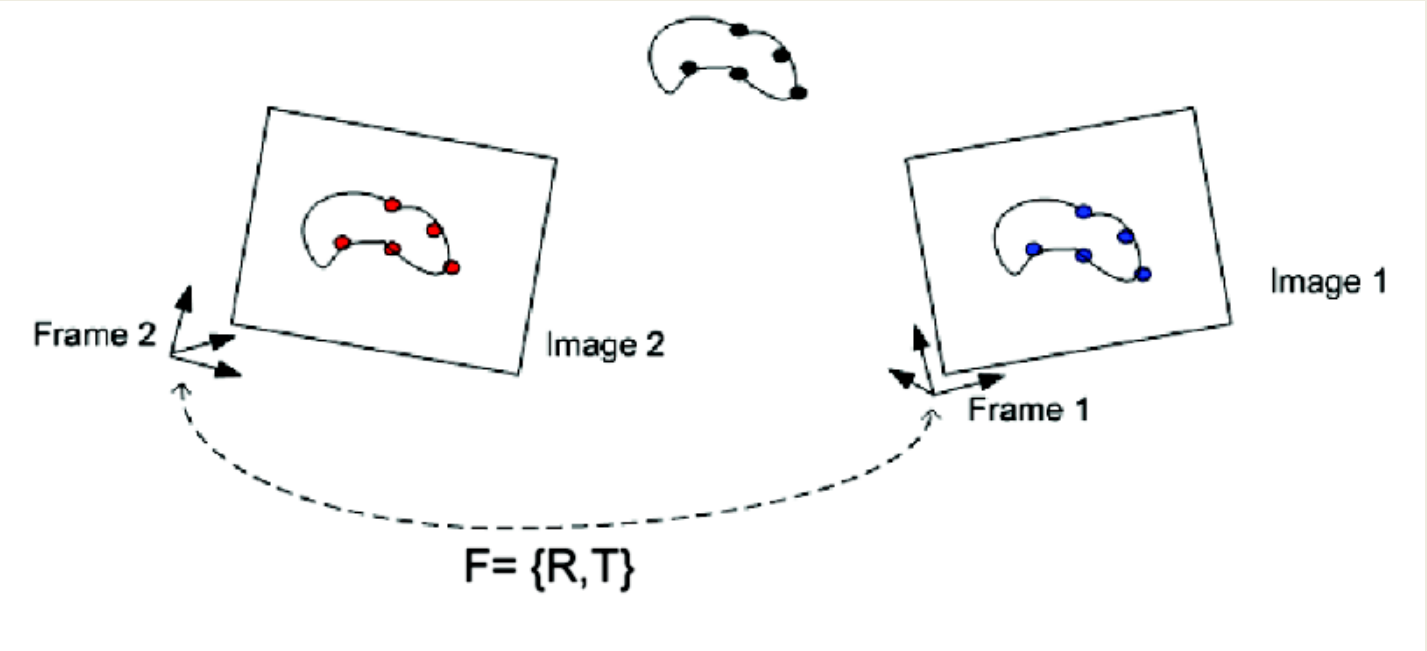
This talk will focus on the control and performance issues, leaving aside the computer vision issues (e.g., feature tracking).

The main issues --- how to choose $s(t)$ and the corresponding control law:

- Using 3D reconstruction to define $s(t)$
- Using image data to directly define $s(t)$
- Partitioning degrees of freedom
- Switching between controllers

Position-Based Visual Servo Control

Position-Based Visual Servo Control



- Computer vision data are used to compute the pose of the camera (d and R) relative to the world frame
- The error $e(t)$ is defined in the pose space $d \in \mathbb{R}^3$, $R \in SO(3)$.
- The control signal $\xi = (v, \omega)$ is a camera body velocity.
- The camera velocity ξ is specified w.r.t. the camera frame.

If the goal pose is given by $d = 0$, $R = I$, the role of the computer vision system is to provide, in real time, a measurement of pose error.

PBVS (cont.)

If $u\theta$ is the axis/angle parameterization of R, the error is given by

$$e(t) = \begin{bmatrix} d \\ u\theta \end{bmatrix}$$

and its derivative is given by

$$\dot{e} = \begin{bmatrix} R & 0 \\ 0 & L_\omega(u\theta) \end{bmatrix} \xi = L_{pbvs}(u\theta)\xi$$

in which [Malis 98]

$$L_\omega(u\theta) = I - \frac{\theta}{2} u_\times + \left(1 - \operatorname{sinc}\theta / \operatorname{sinc}^2 \frac{\theta}{2}\right) u_\times$$

PBVS (cont.)

Since L_ω is nonsingular when $\theta \neq 2k\pi$, [Malis, Chaumette, Boudet 99], to achieve the error dynamics $\dot{e} = -\lambda e$ we can use

$$-\lambda e = \dot{e} = L_{pbvs}(u\theta)\xi \quad \rightarrow \quad \xi = -\lambda \left(L_{pbvs}(u\theta)\right)^{-1} e$$

The motivation: The solution of the differential equation $\dot{e} = -\lambda e$ is a decaying exponential.

That's nice --- but how do we know that it really works? After all, $e(t)$ is a vector, and L, is not a constant matrix...

This isn't really a nice, scalar, first-order linear differential equation.

Lyapunov Theory

Lyapunov theory provides a powerful tool for analyzing the stability of nonlinear systems.

- Consider a nonlinear system on $\mathbb{R}^n$

$$\dot{x} = f(x)$$

where $f(x)$ is a vector field on $\mathbb{R}^n$, and suppose that *f(0) = 0*.

- The origin in $\mathbb{R}^n$ is said to be an **equilibrium point** for the system.

What does this have to do with our visual servo problem?!?!

- If we use a control law $u(t) = \xi = -\lambda L^{+} e$ and if $\dot{e} = L\xi$,
- then $e(t) = 0$ is an equilibrium point for our visual servo system, since

$$\mathrm{e} = 0 \rightarrow -\lambda L^{+} e = \xi = 0 \rightarrow L\xi = \dot{e} = 0$$

When the error is zero, the control input is zero, thus $\dot{e}$ is zero.

Lyapunov Theory

Lyapunov theory provides a powerful tool for analyzing the stability of nonlinear systems.

- Consider a nonlinear system on $\mathbb{R}^n$

$$\dot{x} = f(x)$$

where $f(x)$ is a vector field on $\mathbb{R}^n$, and suppose that *f(0) = 0*.

- The origin in $\mathbb{R}^n$ is said to be an **equilibrium point** for the system.

Lyapunov Functions:

- Let $\mathcal{L}(x)$: $\mathbb{R}^n \rightarrow \mathbb{R}$ be a function with continuous first partial derivatives in a neighborhood of the origin.

- Let $\mathcal{L}$ be positive definite: $\mathcal{L}(0) = 0$, $\mathcal{L}(x) > 0$ for all $x \neq 0$.

- $\mathcal{L}$ is called a **Lyapunov function candidate** for the system.

Lyapunov Theory (cont)

THEOREM: The origin is a ***stable equilibrium*** for the system if there exists a Lyapunov function candidate $\mathcal{L}$ such that $\dot{\mathcal{L}}$ is negative semi-definite along solution trajectories for the system, i.e.,

$$\dot{\mathcal{L}} = \frac{\partial \mathcal{L}}{\partial x}\dot{x} = \frac{\partial \mathcal{L}}{\partial x} f(x) \leq 0$$

THEOREM: The origin is ***asymptotically stable*** if there exists a Lyapunov function candidate $\mathcal{L}$ such that $\dot{\mathcal{L}}$ is negative definite along solution trajectories for the system

$$\dot{\mathcal{L}} = \frac{\partial \mathcal{L}}{\partial x} f(x) < 0$$

Lyapunov Theory and Visual Servo Control

The two versions of stability provide different sorts of performance guarantees:

- Stability guarantees that the system will remain within a neighborhood of the equilibrium point, provided the initial state is sufficiently close to the equilibrium point.
- Asymptotic stability guarantees that the system will converge to the equilibrium point, provided the initial state is sufficiently close to the equilibrium point.

In some cases, the system error is the simplest Lyapunov function candidate --- this is the case for many visual servo systems.

$$\mathcal{L} = \frac{1}{2}\|e(t)\|^2 \quad \rightarrow \quad \dot{\mathcal{L}} = e^T(t)\dot{e}(t)$$

Lyapunov stability of PBVS

Recall our PBVS controller:

$$-\lambda e = \dot{e} = L_{pbvs}(u\theta)\xi \quad \rightarrow \quad \xi = -\lambda \left(L_{pbvs}(u\theta)\right)^{-1} e$$

Using the Lyapunov function $\mathcal{L} = \frac{1}{2}\|e(t)\|^2$ we obtain

$$\dot{\mathcal{L}} = e^T(t)\dot{e}(t)$$

$$= e^T(t)L_{pbvs}(u\theta)\xi$$

Lyapunov stability of PBVS

Recall our PBVS controller:

$$-\lambda e = \dot{e} = L_{pbvs}(u\theta)\xi \quad \rightarrow \quad \xi = -\lambda \left(L_{pbvs}(u\theta)\right)^{-1} e$$

Using the Lyapunov function $\mathcal{L} = \frac{1}{2}\|e(t)\|^2$ we obtain

$$\dot{\mathcal{L}} = e^T(t)\dot{e}(t)$$

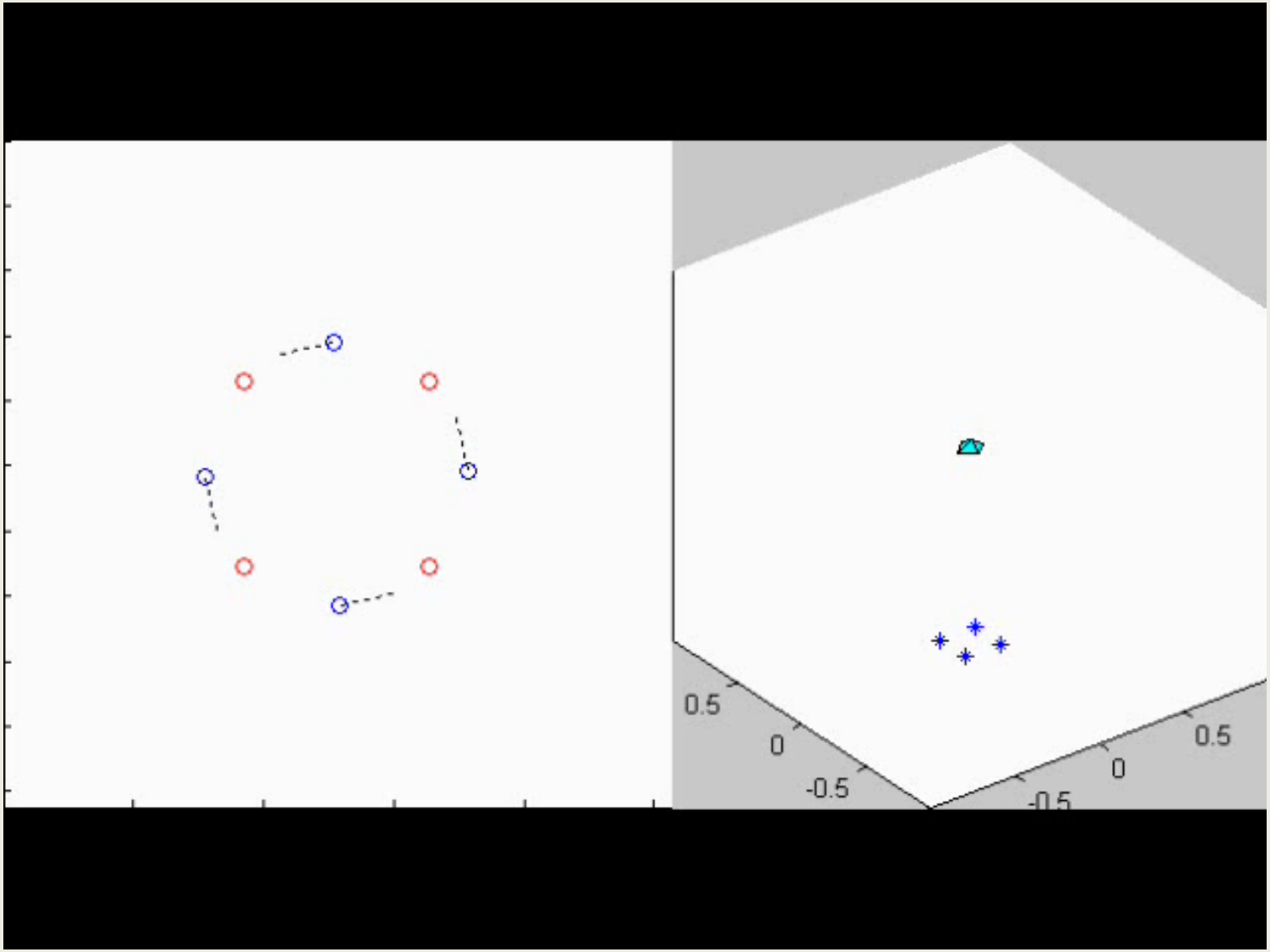
$$= e^T(t)L_{pbvs}(u\theta)\xi$$

$$= -\lambda\, e^T L_{pbvs}(u\theta) \left(L_{pbvs}(u\theta)\right)^{-1} e$$

$$= -\lambda \|e(t)\|^2$$

and we have, not surprisingly, asymptotic stability.

PBVS Example



Why not just use PBVS?

- Feedback is computed using estimated quantities that are a function of the system calibration parameters. Thus,

$$\dot{\mathcal{L}} = -\lambda\, e^T L_{pbvs}(u\theta) \left(\hat{L}_{pbvs}(u\theta)\right)^{-1} e$$

and we need $L_{pbvs}(u\theta) \left(\hat{L}_{pbvs}(u\theta)\right)^{-1}$ to be positive definite.

- Even small errors computing the orientation of the cameras can lead to reconstruction errors that significantly impact system accuracy.
- Position-based control requires an accurate model of the target --- a form of calibration.
- In task space, the robot will move a minimal distance, but in the image, features may move a non-minimal distance during execution.

$\longrightarrow$ Features may leave the field of view.

PBVS Task Example
Large translation and rotation about all axes

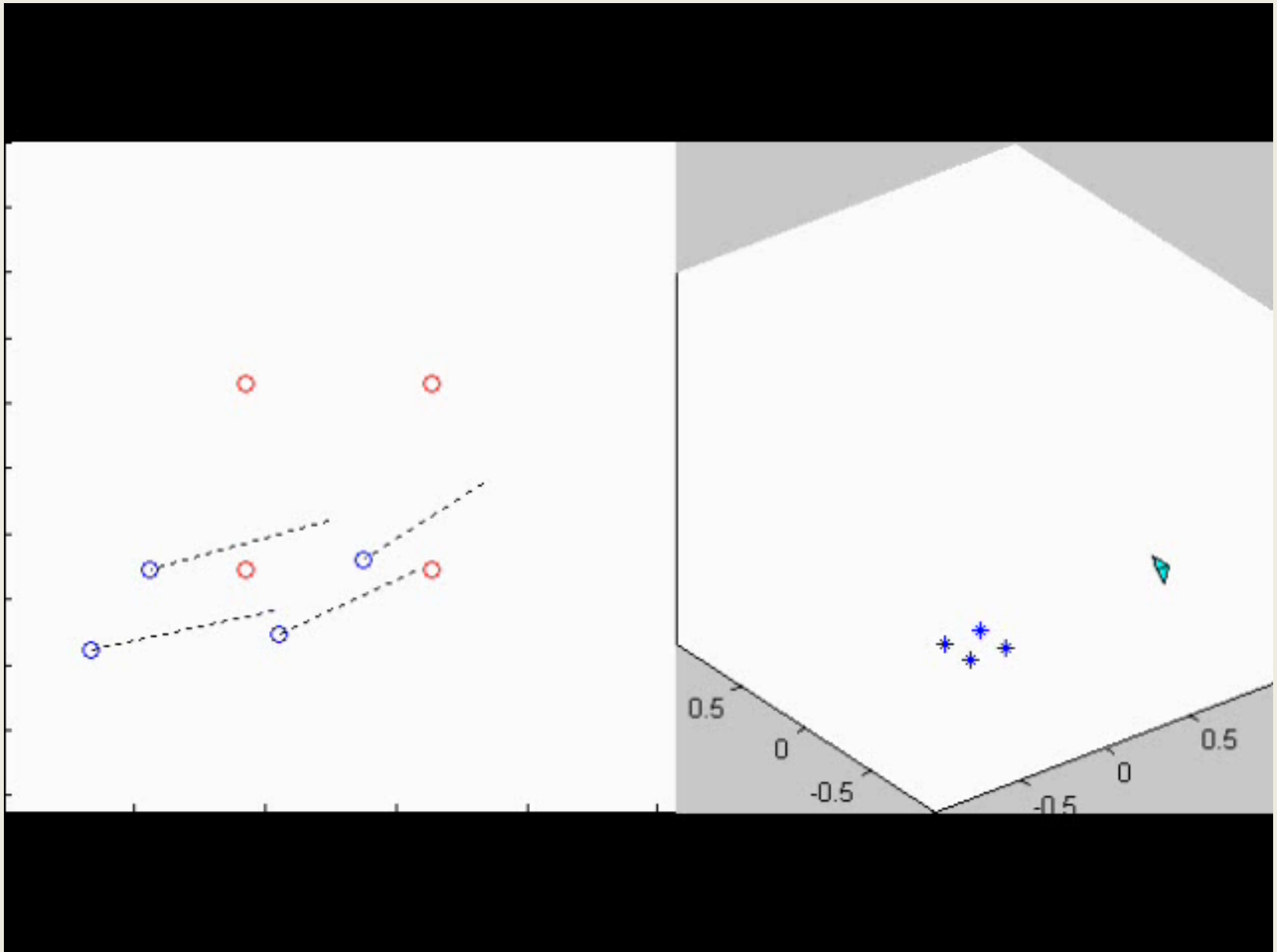


Image-Based Visual Servo Control

Image-Based Visual Servo Control

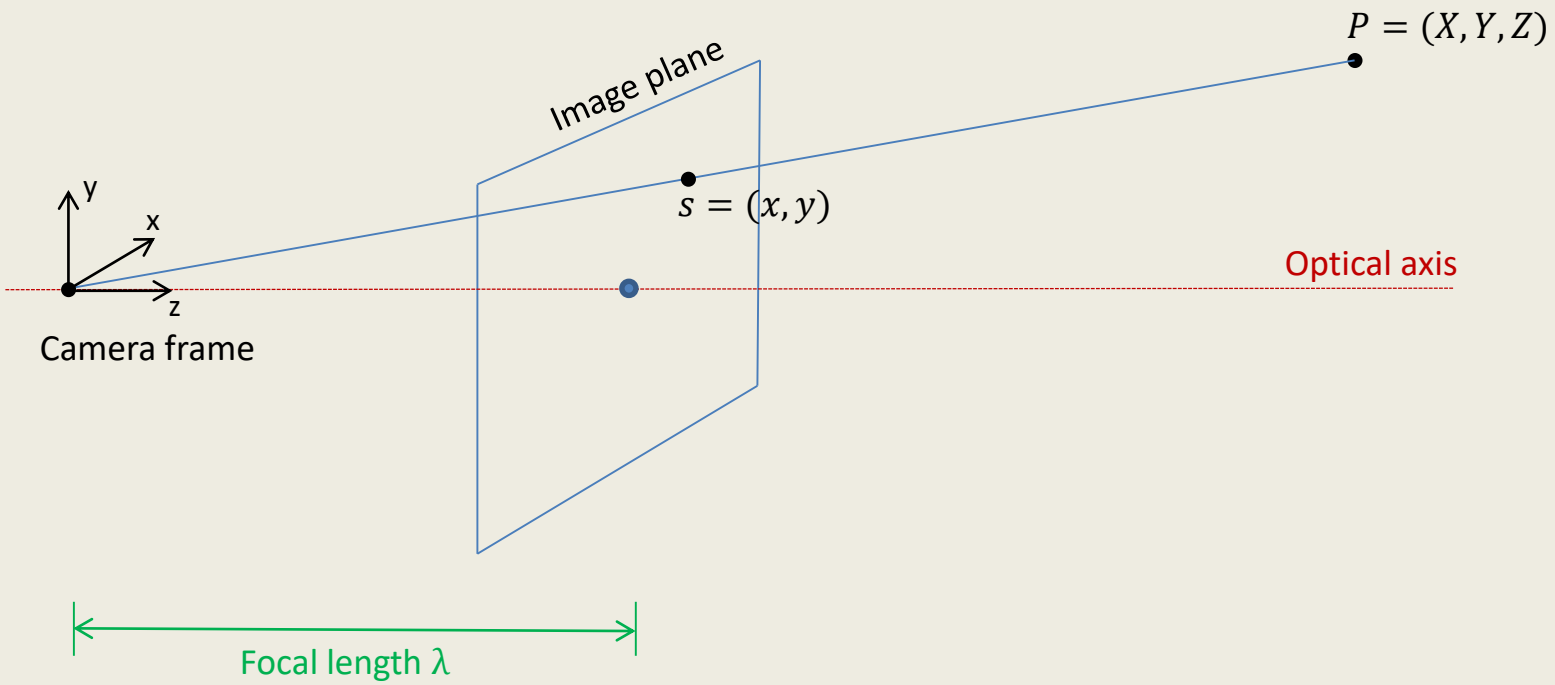
For Image-Based Visual Servo (IBVS)

- Features $s(t)$ are extracted from computer vision data.
- Camera pose is not explicitly computed.
- The error is defined in the image feature space: $e(t) = \ s(t) - s^*$
- The control signal, $\xi = (v, \omega)$ is again a camera body velocity specified w.r.t. the camera frame, but for IBVS it is computed directly using $s(t)$.

For example, if the feature is a single image point with image plane coordinates x and y, we have $s(t) = \ (x(t), y(t))$

Since $\dot{e}(t) = \dot{s}(t)$, we'll need to know the relationship between $\dot{s}$ and ξ to design a controller that achieves the error dynamics $\dot{e} = \ -\lambda e$.

Imaging Geometry



Consider a point P with coordinates (X,Y,Z) w.r.t. the camera frame.

Using perspective projection, P's image plane coordinates are given by

$$x = \lambda \frac{X}{Z} \qquad y = \lambda \frac{Y}{Z}$$

in which λ is the camera focal length.

The Interaction Matrix (for a point feature)

As an example, consider the interaction matrix for a single point, with coordinates (X,Y,Z).

To determine the interaction matrix for the point:

1. Compute the time derivatives for x, y
2. Express these time derivatives in terms of $x, y,\ \dot{X},\ \dot{Y}, \dot{Z}$ and Z
3. Find expressions for $\dot{X},\ \dot{Y}, \dot{Z}$ in terms of ξ and X, Y, Z
   (i.e., eliminate X, Y)
4. Combine equations and grind through the algebra

The Interaction Matrix (for a point feature)

Step 1: Compute the time derivatives for x, y

Recall $x = \lambda \frac{X}{Z}$ and $y = \lambda \frac{Y}{Z}$

Using the quotient rule

$$\dot{x} = \lambda \frac{Z\dot{X} - X\dot{Z}}{Z^2} \quad \text{and} \quad \dot{y} = \lambda \frac{Z\dot{Y} - Y\dot{Z}}{Z^2}$$

The Interaction Matrix (for a point feature)

Step 2: Express time derivatives in terms of $x, y, \dot{X}, \dot{Y}, \dot{Z}, Z$

- The perspective projection equations can be rewritten to give expressions for X and Y as

$$X = \frac{xZ}{\lambda} \quad \text{and} \quad Y = \frac{yZ}{\lambda}$$

- Substitute these into the equations for $\dot{x}$ $\dot{y}$ to obtain

$$\dot{x} = \lambda\frac{\dot{X}}{Z} - \frac{x\dot{Z}}{Z} \quad \text{and} \quad \dot{y} = \lambda\frac{\dot{Y}}{Z} - \frac{y\dot{Z}}{Z}$$

The Interaction Matrix (for a point feature)

Step 3: Find expressions for $\dot{X}, \dot{Y}, \dot{Z}$ in terms of ξ and X, Y, Z

The velocity of (the fixed point) P relative to the camera frame is given by:

$$\dot{P} = -\omega \times P - v$$

which gives equations for each of $\dot{X}, \dot{Y}$ and $\dot{Z}$.

Expanding $\dot{P} = -\omega \times P - v$ we obtain

$$\dot{X} = -v_x - \omega_y Z + \omega_z Y$$
$$\dot{Y} = -v_y - \omega_z X + \omega_x Z$$
$$\dot{Z} = -v_z - \omega_x Y + \omega_y X$$

Now it's just algebra…

The Interaction Matrix (for a point feature)

Step 4: Combine equations and grind through the algebra

Combining equations, we obtain

$$\dot{x} = -\frac{\lambda}{Z} v_x + \frac{x}{Z} v_z + \frac{xy}{\lambda} \omega_x - \frac{(\lambda^2+x^2)}{\lambda} \omega_y + y\omega_z$$

$$\dot{y} = -\frac{\lambda}{Z} v_y + \frac{y}{Z} v_z + \frac{(\lambda^2+y^2)}{\lambda} \omega_x - \frac{xy}{\lambda} \omega_y - x\omega_z$$

These equations can be written nicely in matrix form.

The Interaction Matrix (for a point feature)

In matrix form, we obtain:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix}=\begin{bmatrix}-\frac{\lambda}{Z} & 0 & \frac{x}{Z} & \frac{xy}{\lambda} & -\frac{\lambda^2+x^2}{\lambda} & y\\ 0 & -\frac{\lambda}{Z} & \frac{y}{Z} & \frac{\lambda^2+y^2}{\lambda} & -\frac{xy}{\lambda} & -x\end{bmatrix}\xi$$

The Interaction Matrix (for a point feature)

In matrix form, we obtain:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix}=\begin{bmatrix}-\frac{\lambda}{Z} & 0 & \frac{x}{Z} & \frac{xy}{\lambda} & -\frac{\lambda^2+x^2}{\lambda} & y\\ 0 & -\frac{\lambda}{Z} & \frac{y}{Z} & \frac{\lambda^2+y^2}{\lambda} & -\frac{xy}{\lambda} & -x\end{bmatrix}\xi$$

This can be written more compactly as

$$\dot{s} = L(s,z)\xi$$

The matrix L is known as the *interaction matrix* [Espiau, et al., 1992] or the *image Jacobian*.

Weiss et al. [1987] used *feature sensitivity matrix*, while Feddema et al., [1989] merely used *Jacobian* to describe this matrix.

The Null Space of the Interaction Matrix

The null space of this interaction matrix is spanned by:

$$\begin{bmatrix} x \\ y \\ \lambda \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ x \\ y \\ \lambda \end{bmatrix} \quad \begin{bmatrix} xyZ \\ -(x^2 + \lambda^2)Z \\ \lambda yZ \\ -\lambda^2 \\ 0 \\ x\lambda \end{bmatrix} \quad \begin{bmatrix} \lambda\left(x^2 + y^2 + \lambda^2\right)Z \\ 0 \\ -x\left(x^2 + y^2 + \lambda^2\right)Z \\ xy\lambda \\ -\left(x^2 + \lambda^2\right)Z \\ x\lambda^2 \end{bmatrix}$$

The Null Space of the Interaction Matrix

The null space of this interaction matrix is spanned by:

$$\begin{bmatrix} x \\ y \\ \lambda \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ x \\ y \\ \lambda \end{bmatrix} \quad \begin{bmatrix} xyZ \\ -(x^2 + \lambda^2)Z \\ \lambda yZ \\ -\lambda^2 \\ 0 \\ x\lambda \end{bmatrix} \quad \begin{bmatrix} \lambda(x^2 + y^2 + \lambda^2)Z \\ 0 \\ -x(x^2 + y^2 + \lambda^2)Z \\ xy\lambda \\ -(x^2 + \lambda^2)Z \\ x\lambda^2 \end{bmatrix}$$

Intuitively, this basis of the null space corresponds to

- Translation along a projection ray
- Rotation about a projection ray
- Translation along the camera y-axis, keeping the camera pointed in the correct direction using rotational motions
- Rotation about the camera y-axis, keeping the camera pointed in the correct direction using the linear motion

These are the point motions that cannot be "seen" by the camera.

The Interaction Matrix for Multiple Image Points

- Since $L(s, Z)$ has a nonzero null space, we cannot control all six degrees of freedom for the camera motion using a single image point.
- One solution is to simply use multiple image points.
- In this case, we merely stack the interaction matrices to obtain

$$\dot{s} = \begin{bmatrix} \dot{s}_1(t) \\ \vdots \\ \dot{s}_n(t) \end{bmatrix} = \begin{bmatrix} L_1(s_1, Z_1) \\ \vdots \\ L_n(s_n, Z_n) \end{bmatrix} \xi$$

- Using this approach, three points provide sufficient information to control the camera's six degrees of freedom.
- It is required to know the depth Z_i for each point (or at least an estimate).

Proportional Image-Based Control

As before, to achieve the error dynamics $\dot{e} = -\lambda e$

$$-\lambda e = \dot{e} = \dot{s} = L(s, Z)\xi$$

$$\Longrightarrow \xi = -\lambda\, L^{+}(s, Z)e$$

in which $L^{+} = (L^{T}L)^{-1}L^{T}$.

Using the Lyapunov function $\mathcal{L} = \frac{1}{2}\|e\|^{2}$ we obtain

$$\begin{aligned}\dot{\mathcal{L}} &= e^{T}\,\dot{e} \\ &= e^{T}L\xi \\ &= -\lambda e^{T}LL^{+}e\end{aligned}$$

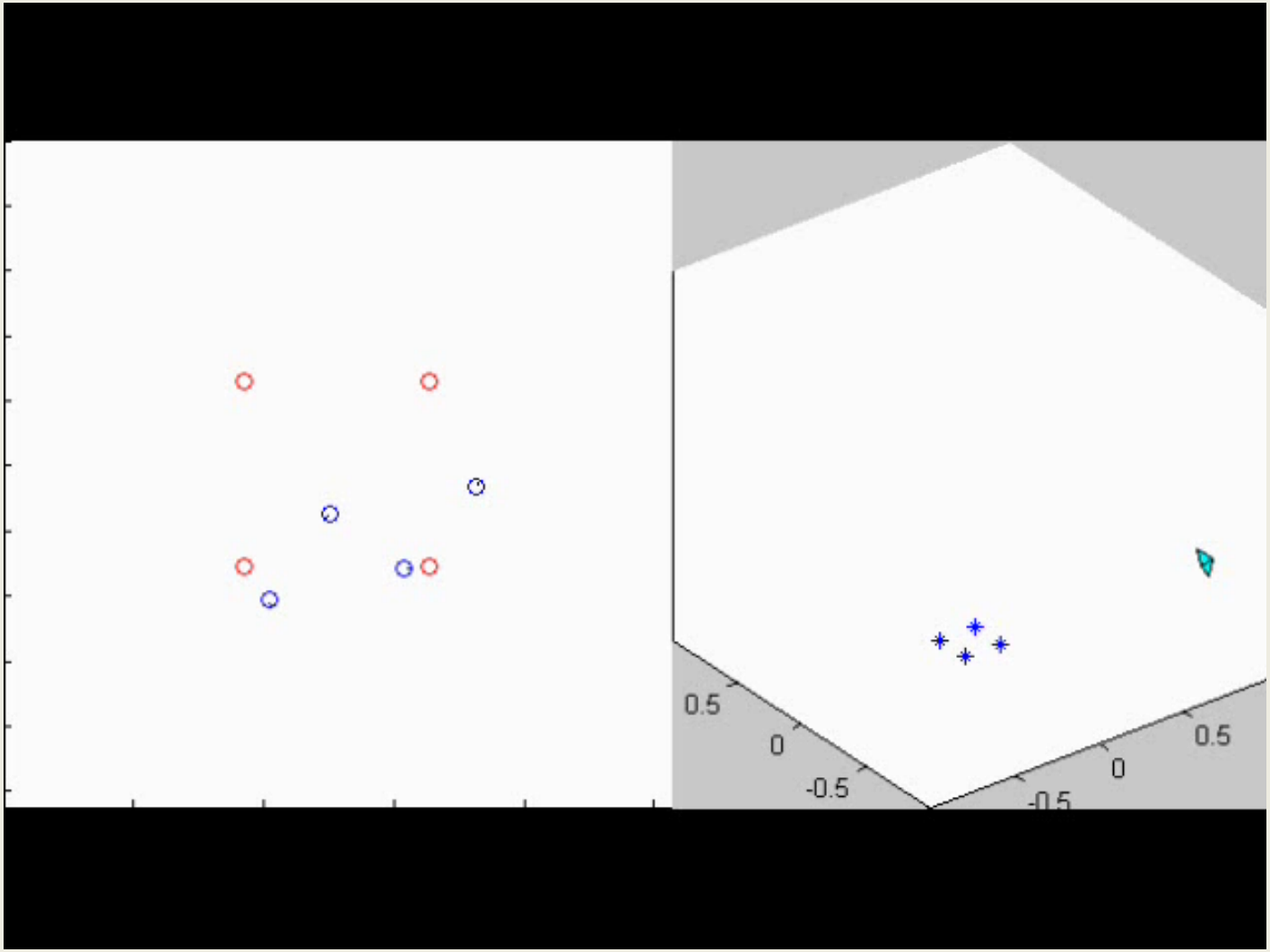
We have asymptotic stability when the matrix LL^{+} is positive definite.

Unfortunately, this condition is rarely achieved, e.g., when $\dim(s) > 6$.

More on this a bit later…

IBVS Task Example

Large Translation and Rotation About All Axes



Direct Estimation of the Interaction Matrix

- Methods to numerically estimate the interaction matrix rely on observation of a variation of the features Δs due to a known or measured camera motion $\Delta\xi$ and these are related by

$$L_s \Delta\xi = \Delta s$$

which provides k equations, while we have $k \times 6$ unknowns in L_s.

- Using a set of N independent camera motions, with $N > 6$, we can estimate L_s by solving

$$L_s A = B$$

where the columns of $A \in \mathbb{R}^{6\times N}$ and $\mathrm{B} \in \mathbb{R}^{k\times N}$ are formed with the set of camera motions and feature variations.

- The least squares solution is, as usual, given by

$$\widehat{L_s} = BA^{+}$$

Direct Estimation (cont)

- It is possible to estimate directly the numerical value of L^+, which typically provides better behavior.
- The basic relation is, $L^+ \Delta s = \Delta \xi$, which provides 6 equations.
- Using a set of N independent camera motions, with $N > k$, we can estimate L^+ by solving

$$\widehat{L^+} = [\Delta \xi_1 \dots \Delta \xi_N][\Delta s_1 \dots \Delta s_N]^+$$

in which, $[\Delta \xi_1 \dots \Delta \xi_N] \in \mathbb{R}^{6 \times N}$ and $[\Delta s_1 \dots \Delta s_N]^+ \in \mathbb{R}^{N \times k}$

- In this case, the k columns of $\widehat{L^+}$ are estimated by solving k linear systems, while for the previous case (i.e., solving for $\widehat{L_s}$) the six columns of $\widehat{L_s}$ are estimated by solving six linear systems.

Direct Estimation (cont)

- Optimization methods can also be used to estimate $\widehat{L_s}$.
- These methods typically discretize the system equation, and use an iterative updating scheme to refine the estimate $\widehat{L_s}$ at each stage.
- One such on-line iterative formulation uses the Broyden update rule:

$$\widehat{L_s}(\text{t+1}) = \widehat{L_s}(t) + \frac{\alpha}{\Delta\xi^T \Delta\xi}\left(\Delta s - \widehat{L_s}(t)\Delta\xi\right)\Delta\xi^T$$

in which α defines the update speed.

- The matrix $\left(\Delta s - \widehat{L_s}(t)\Delta\xi\right)\Delta\xi^T$ is of rank one, and such methods are often referred to as rank-one update schemes.
- This method updates the estimate $\widehat{L_s}$ at each iteration.

Direct Estimation (cont)

There are advantages and disadvantages to estimating L_s using these methods.

- Using such numerical estimations in the control scheme avoids all the modeling and calibration steps.
- These methods are particularly useful when using features whose interaction matrix is not available in analytical form. For instance, the main eigenvalues of the Principal Component Analysis of an image have been considered in a visual servoing scheme.
- The drawbacks of these methods is that no theoretical stability and robustness analysis can be made.

Why not use IBVS

Image-based approaches have some nice properties:

- No pose estimation is required, which can save significant computation.
- IBVS is remarkably robust to errors in calibration.
- Even though depth estimates are required, these often enter the closed-loop dynamics as gains, and thus uncertainties can be dealt with by choosing conservative control gains.

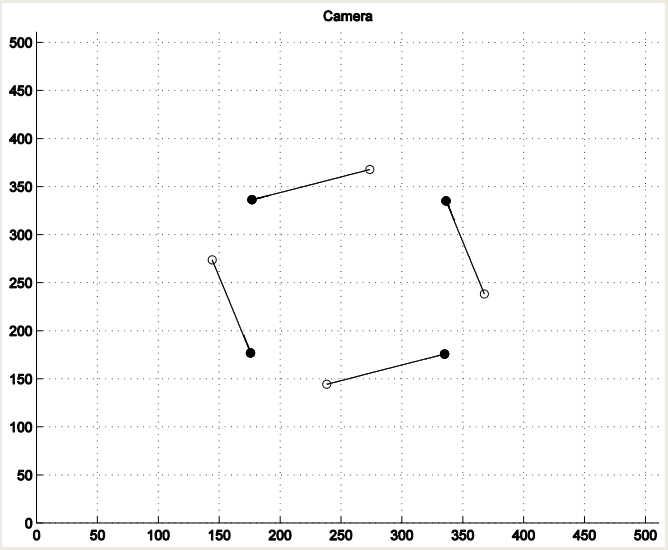
BUT… there are some problems:

- There may be singularities in the interaction matrix.
- Some 3D information is still required (e.g., Z), and must be estimated.
- Unpredictable, often suboptimal Cartesian camera trajectories can occur.

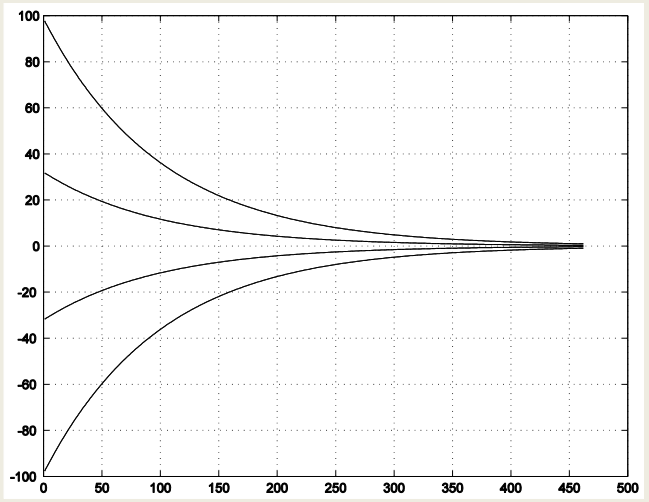
The last of these can cause fairly serious problems.

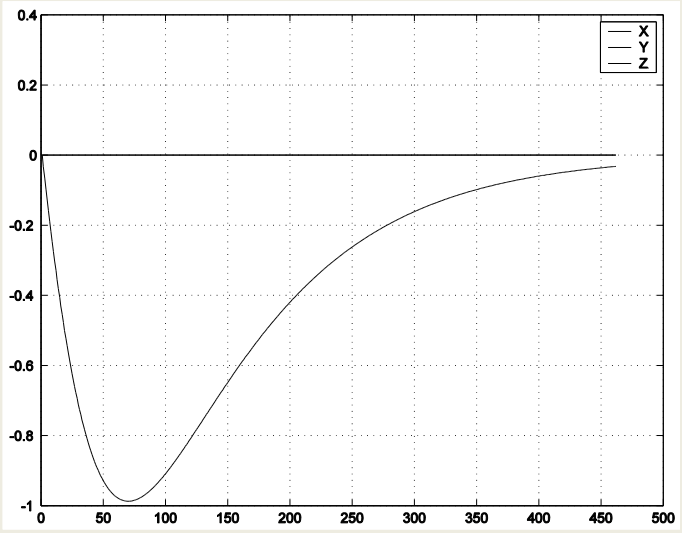
A small rotation about the optic axis

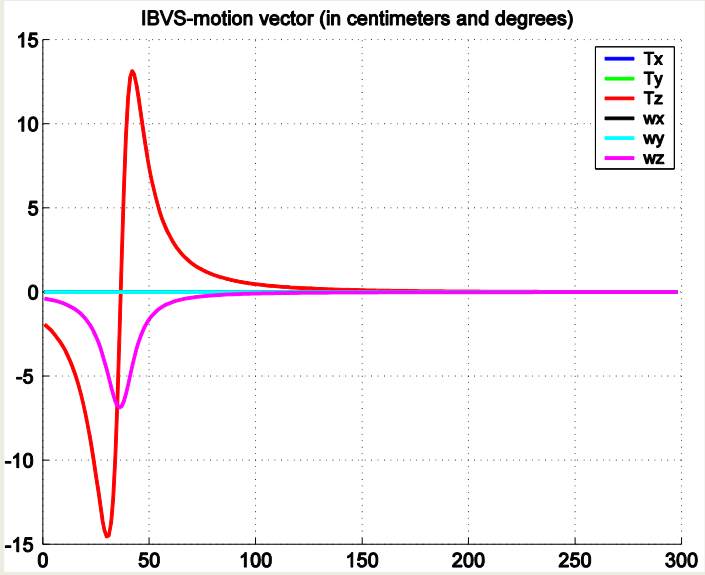
Image Motion



Feature Error





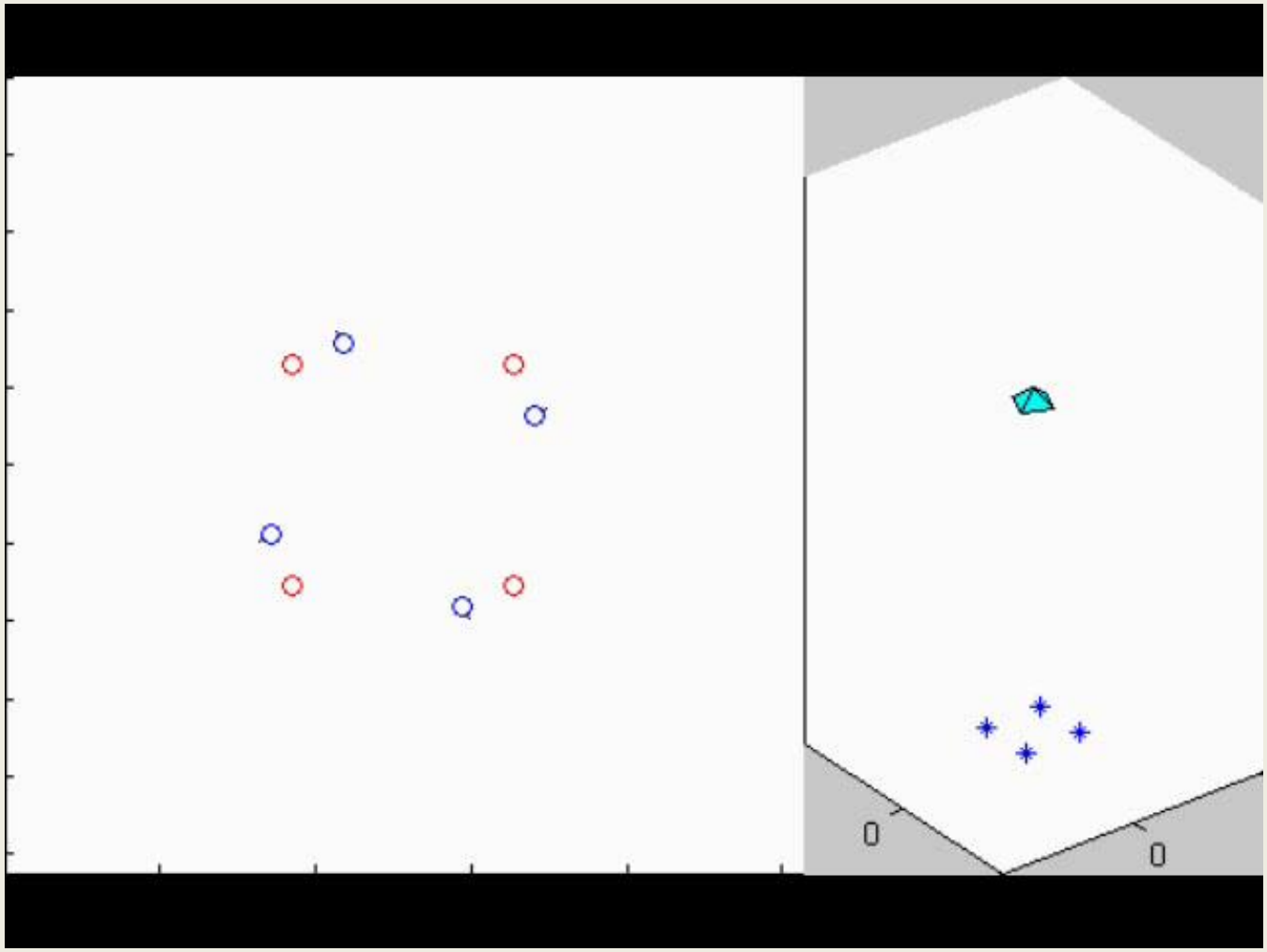


X,Y,Z Camera Position

Components of ξ

IBVS Example

160° rotation about the optical axis



The Chaumette Conundrum

A particularly serious example of this problem was demonstrated by Chaumette [1998].

When the required motion is a rotation by π radians about the optic axis, the camera retreats to infinity under traditional IBVS control.

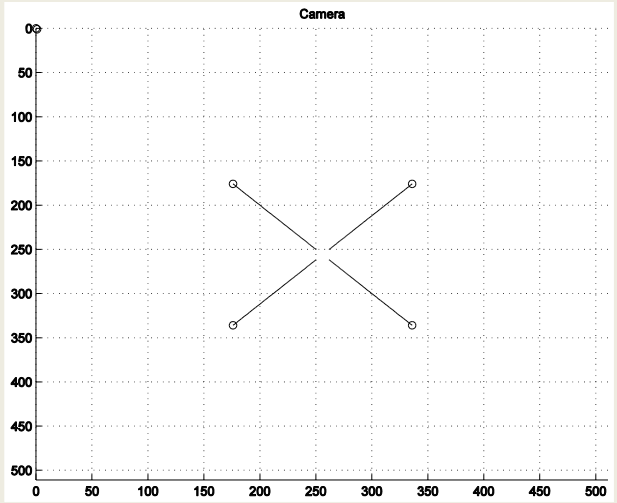
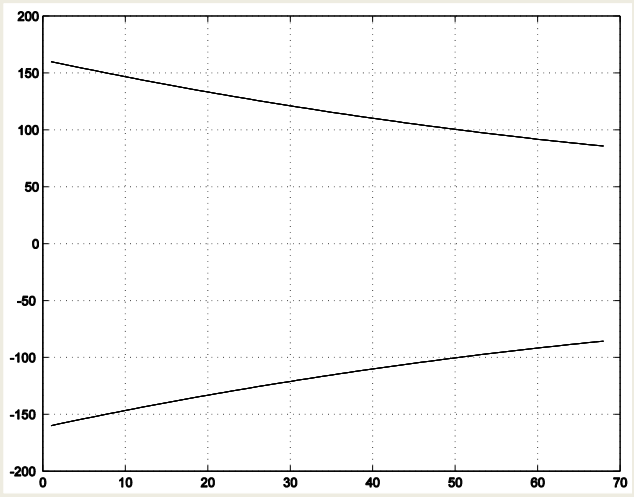
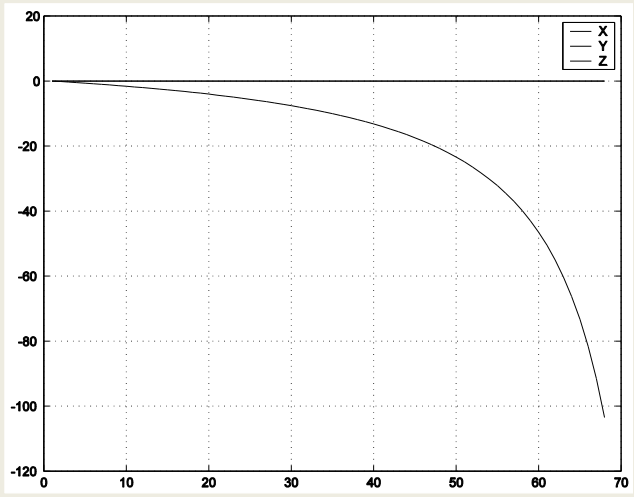


Image-plane feature trajectories



Feature error trajectories



Cartesian camera translation

Coping with PBVS and IBVS performance problems

It seems that neither PBVS nor IBVS are ideal choices for visual servo control.

There are at least two ways to cope with these problems:

- Partition the system's degrees of freedom, controlling some with IBVS, some with PBVS, some with other new methods.
- Partition the system along the time axis, switching between IBVS, PBVS (or other) controllers.

We'll have a look at both of these now…