Robotics, Vision, & Control

Xu Chen

Sp 2021

Motivation

— Classical industrial robot-object interaction:

- need to know the object pose : place the object precisely with, e.g. expensive mechanical jigs & fixtures
- need to assure robots achieve the target pose : tune the robot precisely for each task

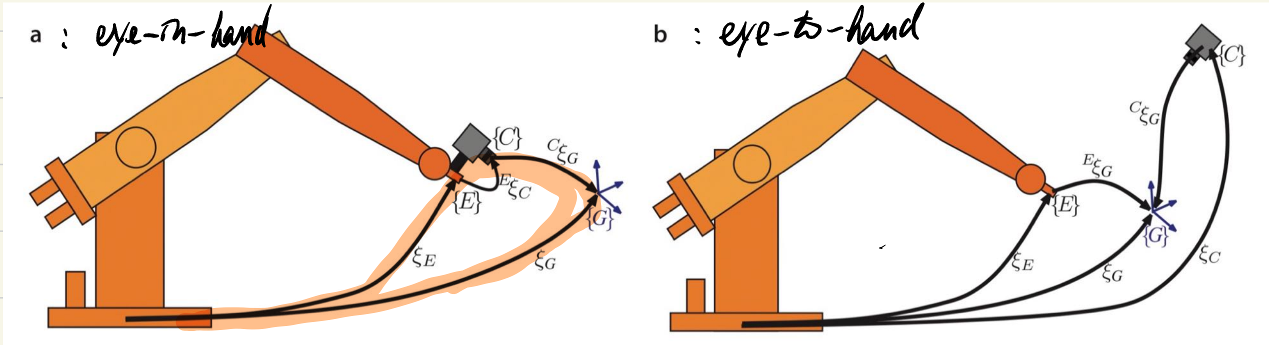
⇒ a heavy & stiff robot demanding powerful actuators & high-quality sensors & sophisticated controllers, all at a high cost.

— Root cause of the issue : the robot doesn't know how it is doing

— vision-based control of robots : robot moves towards the observed object wherever it may be in the workspace.

+ part position tolerance can be relaxed.
+ comes "for free" the ability to deal with moving parts
+ errors in the intrinsic accuracy of the robot will be compensated for.

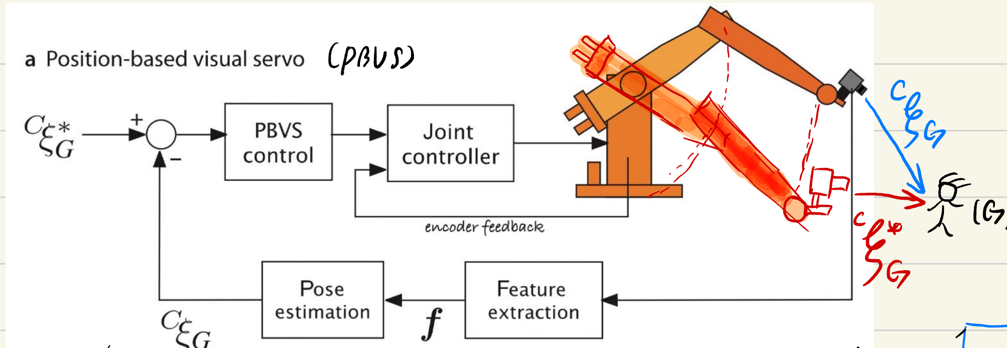
Sensor Configuration:



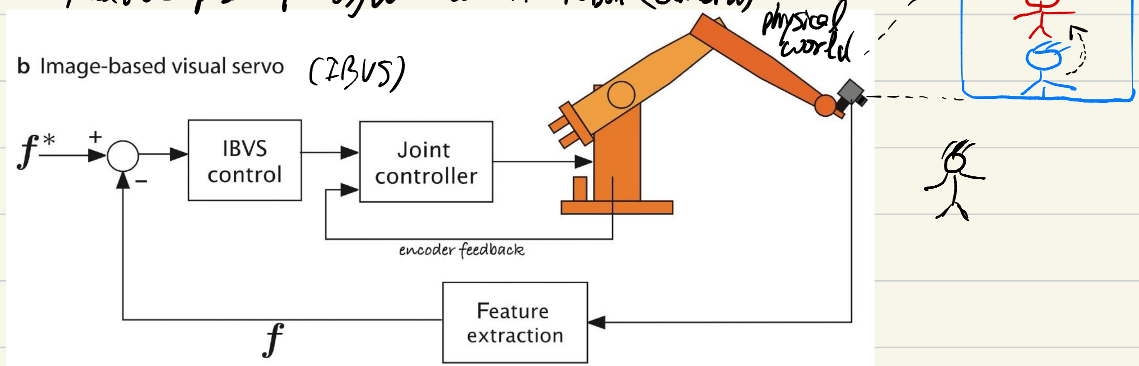
- camera carried by robot
- end-point closed-loop configuration

- camera fixed in the world
- end-point open-loop configuration

Control Configuration

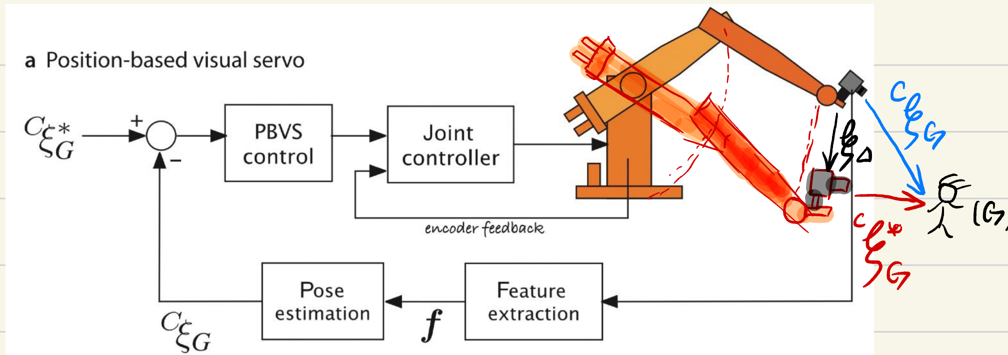


PBVS: focus on relative pose of object w.r.t. robot (camera) in the physical world



IBVS: focus on the position of object in image plane.

Position-based visual servo



Requires: object geometry, camera parameters, image features

Goal: move robot pose from current $^C\xi_G$ to target $^C\xi^*_G$

$\Leftarrow$ required camera motion $\xi_\Delta = {}^C\xi_G \ominus {}^C\xi^*_G$

↑ from pose estimation

MATLAB

$\xi_\Delta = {}^CT_G \left({}^CT^*_G\right)^{-1}$

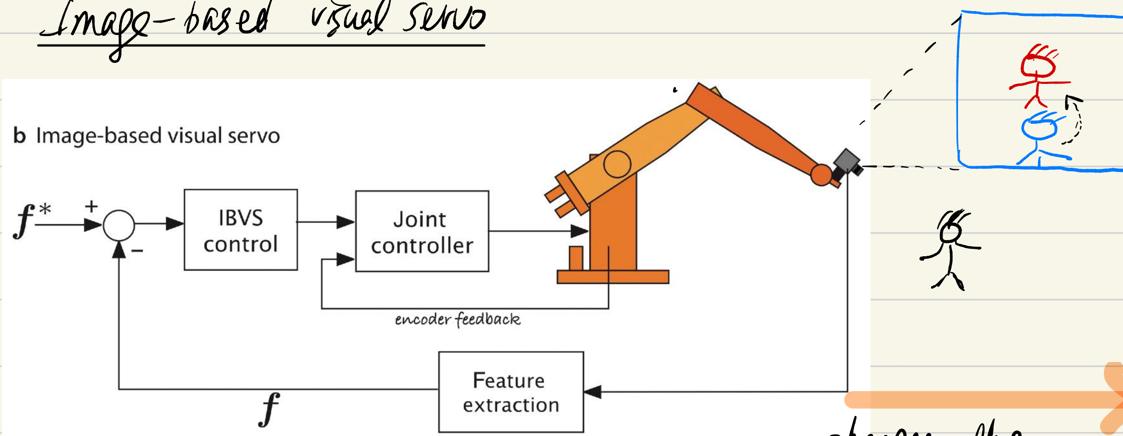
Iterative solution:

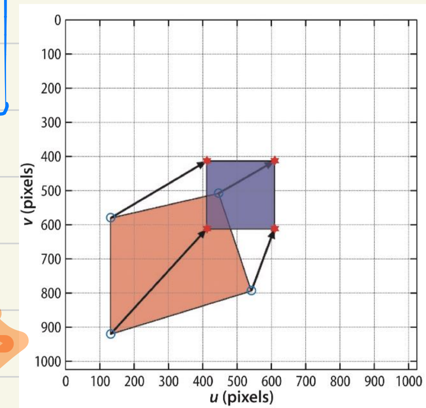
$$\xi_C[k+1] = \xi_C[k] \oplus \lambda \underbrace{\xi_\Delta[k]} \qquad 0<\lambda<1$$

$\xi_C[k+1] = T_C[k] \cdot \lambda T_\Delta[k]$

at each step,
do pose estimation $^C\xi_G[k]$ & compute $\xi_\Delta[k]$

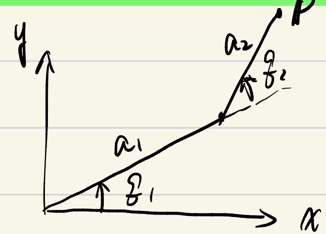
Image-based visual servo





changes the problem from pose estimation to control of points in the projected image plane (pose & kinematics are implicitly changed)

Recall 1: Jacobian & joint space → Cartesian Space



forward kinematics:

$$p = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} a_2\cos(q_1+q_2) + a_1\cos q_1 \\ a_2\sin(q_1+q_2) + a_1\sin q_1 \end{bmatrix}$$

derivative of p:

$$\frac{dp}{dt} = \underbrace{\frac{dp}{dq}}_{\triangleq J(q)} \frac{dq}{dt} \quad \text{i.e. } \dot{p} = J(q)\dot{q}$$

$$\triangleq J(q) = \begin{pmatrix} \frac{dp_1}{dq_1} & \frac{dp_1}{dq_2} \\ \frac{dp_2}{dq_1} & \frac{dp_2}{dq_2} \end{pmatrix}$$

$$= \begin{pmatrix} -a_2\sin(q_1+q_2) - a_1\sin q_1 & -a_2\sin(q_1+q_2) \\ a_2\cos(q_1+q_2) + a_1\cos q_1 & a_2\cos(q_1+q_2) \end{pmatrix}$$

$J(q)$: called the Jacobian matrix.
maps velocity from the joint space to the end-effector's Cartesian space

Generalization to 3D case

Now with rotation & translation

6x6 Jacobian matrix (aka the geometric Jacobian)

$$ {}^{0}\nu = \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \frac{dp}{dt} = {}^{0}J(q)\,\dot{q} $$

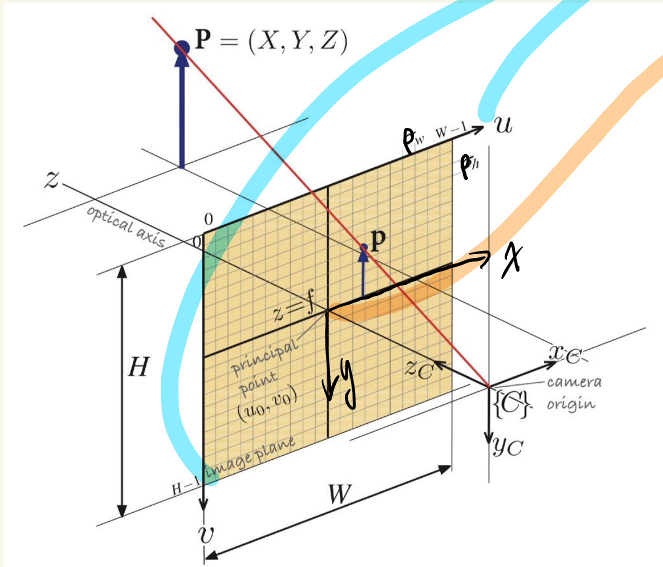
defines the <u>instantaneous forward kinematics</u>

spatial velocity of the end-effector in the world coordinate

MATLAB: p560.jacob0(qn)

${}^{0}J(q)$: maps joint velocity to end-effector spatial velocity on the world coordinate frame

Recall: Perspective Projection



pixel axis: index u & v are nonnegative

principal point: geometric center (u_0, v_0)

$\Rightarrow$ coordinates in pixel domain:

$$u = \frac{x}{p_w} + u_0 \qquad v = \frac{y}{p_h} + v_0$$

pixel width   pixel height

Perspective projection: $x = \frac{f}{Z} \cdot X \qquad y = \frac{f}{Z} Y$

$$\Rightarrow \boxed{u = \frac{f}{p_w Z} X + u_0 \qquad v = \frac{f}{p_h Z} Y + v_0}$$

Moving point in Image Plane

Points have motion in image plane in ibvs.

General velocity formula:

(velocity of $P = (X, Y, Z)$ relative to the camera)

translation

$$\dot{P} = -\omega \times P - \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix}$$

$$= - \begin{vmatrix} i & j & k \\ \omega_x & \omega_y & \omega_z \\ X & Y & Z \end{vmatrix} - \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix}$$

$$\Rightarrow \begin{cases} \dot{X} = \omega_z Y - \omega_y Z - V_x \\ \dot{Y} = Z\omega_x - X\omega_z - V_y \\ \dot{Z} = X\omega_y - Y\omega_x - V_z \end{cases}$$

On the other hand:

$$\boxed{u = \frac{f}{\rho_w Z} X + u_0 \qquad v = \frac{f}{\rho_h Z} Y + v_0}$$

define

$\bar{u} = u - u_0$

$\bar{v} = v - v_0$

$$\Longrightarrow \begin{cases} \dot{\bar{u}} = \dfrac{f}{\rho_w} \dfrac{\dot{X}Z - X\dot{Z}}{Z^2} \\ \dot{\bar{v}} = \dfrac{f}{\rho_h} \dfrac{\dot{Y}Z - Y\dot{Z}}{Z^2} \end{cases}$$

$$\Rightarrow \begin{bmatrix} \dot{\bar{u}} \\ \dot{\bar{v}} \end{bmatrix} = \begin{bmatrix} -\dfrac{f}{\rho_w Z} & 0 & \dfrac{\bar{u}}{Z} & \dfrac{\rho_h \bar{u}\bar{v}}{f} & -\dfrac{f^2 + \rho_w^2 \bar{u}^2}{\rho_w f} & \dfrac{\rho_h \bar{v}}{\rho_w} \\ 0 & -\dfrac{f}{\rho_h Z} & \dfrac{\bar{v}}{Z} & \dfrac{f^2 + \rho_h^2 \bar{v}^2}{\rho_h f} & -\dfrac{\rho_w \bar{u}\bar{v}}{f} & -\dfrac{\rho_w \bar{u}}{\rho_h} \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

Moving Point in Image Plane

velocity in image plane

camera velocity in Cartesian space

$$\begin{bmatrix} \dot{\bar u} \\ \dot{\bar v} \end{bmatrix} = \begin{bmatrix} -\frac{f}{\rho_w Z} & 0 & \frac{\bar u}{Z} & \frac{\rho_h \bar u \bar v}{f} & -\frac{f^2+\rho_w \bar u^2}{\rho_w f} & \frac{\rho_h \bar v}{\rho_w} \\ 0 & -\frac{f}{\rho_h Z} & \frac{\bar v}{Z} & \frac{f^2+\rho_h^2 \bar v^2}{\rho_h f} & -\frac{\rho_w \bar u \bar v}{f} & -\frac{\rho_w \bar u}{\rho_h} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ w_x \\ w_y \\ w_z \end{bmatrix} \xrightarrow[\text{\& } f' \triangleq \frac{f}{\rho}]{\text{if } \rho_w = \rho_h = \rho} \begin{bmatrix} \dot{\bar u} \\ \dot{\bar v} \end{bmatrix} = \underbrace{\begin{bmatrix} -\frac{f'}{Z} & 0 & \frac{\bar u}{Z} & \frac{\bar u \bar v}{f'} & -\frac{f'^2+\bar u^2}{f'} & \bar v \\ 0 & -\frac{f'}{Z} & \frac{\bar v}{Z} & \frac{f'^2+\bar v^2}{f'} & -\frac{\bar u \bar v}{f'} & -\bar u \end{bmatrix}} \begin{bmatrix} v_x \\ v_y \\ v_z \\ w_x \\ w_y \\ w_z \end{bmatrix}$$

$$\dot p_{2\times1} = \underline{\underline{J_p(p, Z)}}\, \nu_{6\times1}$$

2x6 image Jacobian aka feature sensitivity matrix

But we want to control robot at some ν to achieve a desired $\dot p$

try $\nu = J_p(p, Z)^{-1} \dot p$ impossible b/c $J_p \in \mathbb{R}^{2\times6}$

do instead:

$$\nu = \begin{bmatrix} J_p(p_1, Z_1) \\ J_p(p_2, Z_2) \\ J_p(p_3, Z_3) \end{bmatrix}^{-1}_{6\times6} \begin{bmatrix} \dot{\bar u}_1 \\ \dot{\bar v}_1 \\ \dot{\bar u}_2 \\ \dot{\bar v}_2 \\ \dot{\bar u}_3 \\ \dot{\bar v}_3 \end{bmatrix}$$

matrix invertible as long as the points are not coincident or collinear.

The image Jacobian $J_p(p, Z)$

$$\dot{p}_{2\times1} = J_p(p, Z)\, \nu_{6\times1}$$

image plane — camera motion in physical space

- $J_p(p, Z)$ has a null space of $\mathbb{R}^{6\times4}$, representing camera velocities that cause no motion of the point on the image.

if $\nu^0 \in Null(J_p)$

$\Rightarrow J_p \nu^0 = 0$

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

e.g.

```
>> cam = CentralCamera('default')      % load camera model
>> cam.pp                              % principal point
>> J = cam.visjac_p(cam.pp', 1)
```

depth Z

```
>> null(J)
```

Controlling Feature Motion

What camera motion is needed in order to move the image features at a desired velocity?

3-point case:

$$\nu = \begin{bmatrix} J_p(p_1, Z_1) \\ J_p(p_2, Z_2) \\ J_p(p_3, Z_3) \end{bmatrix}^{-1} \underbrace{\begin{bmatrix} \dot{p}_1 \\ \dot{p}_2 \\ \dot{p}_3 \end{bmatrix}}_{\text{feature velocity}} \qquad \dot{p}_1 = \begin{bmatrix} \dot{u}_1 \\ \dot{v}_1 \end{bmatrix}$$

$$= \begin{bmatrix} J_p(p_1, Z_1) \\ J_p(p_2, Z_2) \\ J_p(p_3, Z_3) \end{bmatrix}^{-1} \lambda (p^* - p)$$

feature velocity : design parameter

e.g. first-order control

$$\dot{p} = \lambda (p^* - p)$$

$$\Rightarrow \underset{\text{desired}}{p^*} \longrightarrow \boxed{\frac{\lambda}{s+\lambda}} \longrightarrow p$$

* DC gain : 1
* time constant : $\frac{1}{\lambda}$
* Exponential convergence

Benefit: we never needed to compute camera poses!

N-point case:

$$\nu = \begin{bmatrix} J_p(p_1, Z_1) \\ J_p(p_2, Z_2) \\ \vdots \\ J_p(p_N, Z_N) \end{bmatrix}^{\dagger} \lambda (p^* - p)$$

← psudo inverse

$$A^{\dagger} = (A^T A)^{-1} A^T$$

A few more remarks

- $J(p, Z)$: needs Z, the depth of the point & camera intrinsic parameters

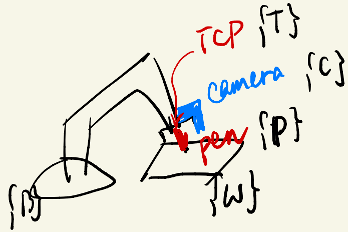
  In practice, ibvs is remarkably tolerant to errors in Z, can be assumed constant, or estimated

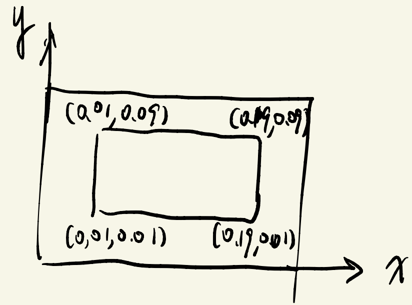
  from camera calibration

- besides using points as features, line/circle/planar features can also be used.

- in-class demo: sl_ibvs & custom code.

Design Exercise





$${}^{B}\xi_{T} = ?$$

$${}^{C}\xi_{W} = ?$$

$${}^{B}\xi_{W} = {}^{B}\xi_{T} \oplus {}^{T}\xi_{C} \oplus {}^{C}\xi_{W}$$

$$\left({}^{B}T_{W} = {}^{B}T_{T} \cdot {}^{T}T_{C} \cdot {}^{C}T_{W}\right)$$