GMAR Robotics Summer School 2021

VISUAL SERVOING
— for —
NAVIGATION & MANIPULATION

Antonio Paolillo

Dalle Molle Institute for Artificial Intelligence (IDSIA), USI-SUPSI
Lugano, Switzerland

◇

Innsbruck - August 25, 2021

Outline

- What is visual servoing?
- Why do we need visual servoing?
- How do we build visual servoing?
- What can we do with visual servoing?

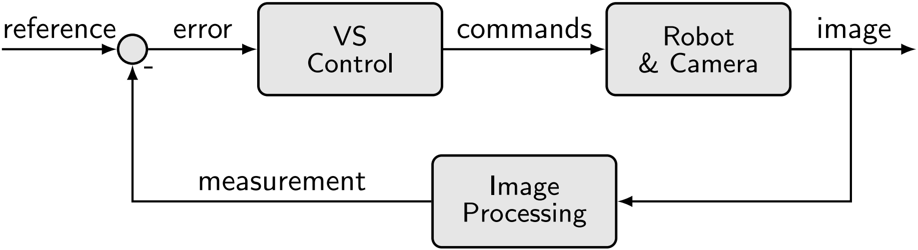
Definition of Visual Servoing (VS)

“ VS is the use of computer *vision* data in the servo loop that *controls* the motion of a robot ”

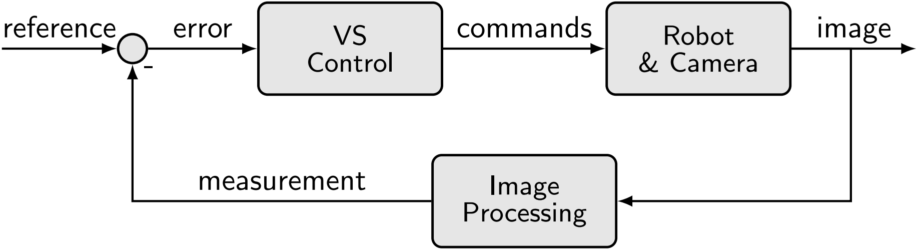
“ VS is the action taken by a *vision-based control* ”

“ VS is the way to provide a control algorithm with *visual feedback* to reach a desired target ”

Block diagram & scheme classification



Block diagram & scheme classification

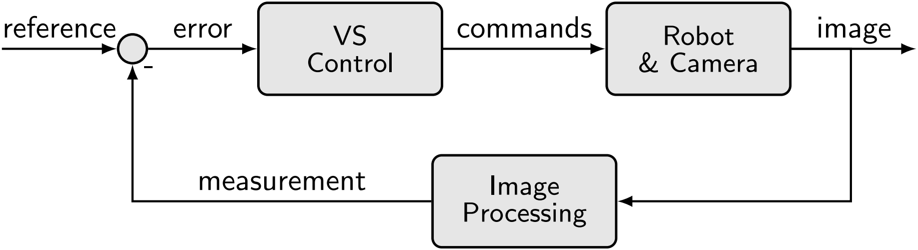


Two main VS schemes:

1. Position-based visual servoing (*PBVS*)
   – More complicated image processing (need to reconstruct a pose)
   + Relatively easier control law
2. Image-based visual servoing (*IBVS*)
   + Easier image processing (it is a features extraction)
   – More complicated control law

▶ Other options are also possible, such as *2.5D VS*

IBVS block diagram



Most of the lecture focuses on *IBVS*

reference	desired visual features	$\mathbf{s}^*$
measurement	current visual features	$\mathbf{s}$
error	visual error	$\mathbf{e} = \mathbf{s} - \mathbf{s}^*$
commands	velocities command	$\mathbf{v}$

Definition of visual feature

- In computer vision, it is the set of pixels for which the *link* between photometric measurement and geometric primitives can be established
- It is the attempt to *summarize* the richness of data coming from the camera video stream
  - Be aware of the *information loss* that this “summary” involves
- It is the *gist* of the scene needed to control the robot
- It is the summary information got from the captured image, needed to close the VS loop and achieve a desired robotic behavior

Why visual features?

Consider the task of looking at the red object

captured image
240×320 *pixels*

Why visual features?

Consider the task of looking at the red object

captured image

240×320 *pixels*

$$\begin{bmatrix} \begin{bmatrix}157\\182\\202\end{bmatrix} & \begin{bmatrix}157\\182\\202\end{bmatrix} & \begin{bmatrix}157\\182\\202\end{bmatrix} & \cdots & \begin{bmatrix}82\\106\\130\end{bmatrix} \\ \begin{bmatrix}157\\182\\202\end{bmatrix} & \begin{bmatrix}157\\182\\202\end{bmatrix} & \begin{bmatrix}157\\182\\202\end{bmatrix} & \cdots & \begin{bmatrix}82\\106\\130\end{bmatrix} \\ & \vdots & & \ddots & \vdots \\ \begin{bmatrix}51\\61\\71\end{bmatrix} & \begin{bmatrix}51\\61\\71\end{bmatrix} & \begin{bmatrix}50\\60\\70\end{bmatrix} & \cdots & \begin{bmatrix}29\\41\\53\end{bmatrix} \end{bmatrix}$$

how it looks like in the PC

$240 \times 320 \times 3$ *matrix of numbers*

Why visual features?

Consider the task of looking at the red object

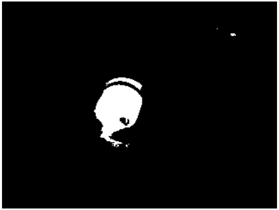
captured image
240 × 320 *pixels*

coordinates of the object centroid
2 *scalar numbers*

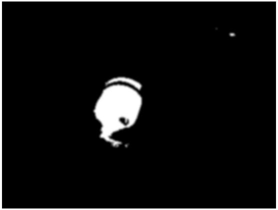
An example of image processing algorithm

- Computer vision community provides many *ready-to-use* tools

original image

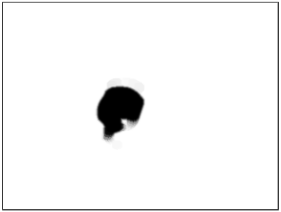


color filtering



smoothing

erode & dilate

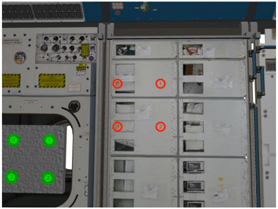


inversion

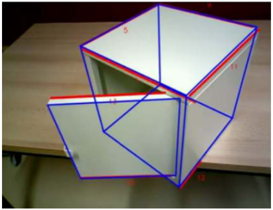
blob detection

- All these operations are available in the *opencv library*, for example

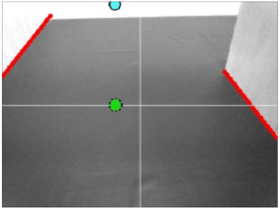
Examples of visual features



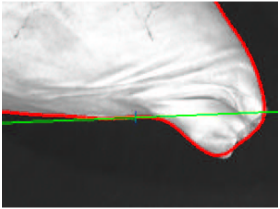
points



lines



reconstructed points



countours

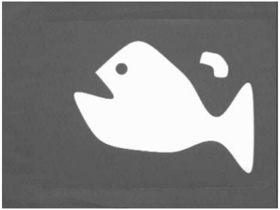
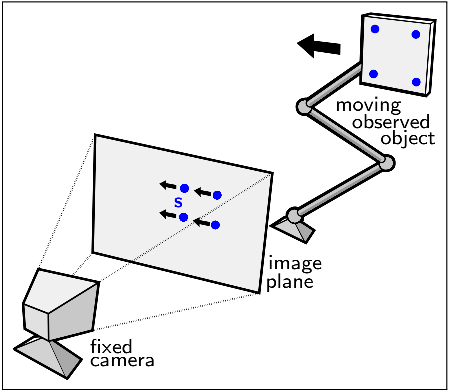


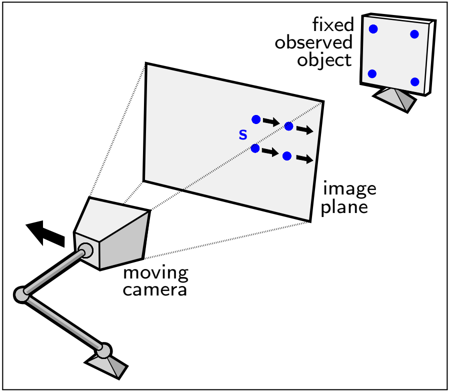
image moments

pixel luminance

In this lecture we focus on *point visual features*

Eye-to-hand & eye-in-hand configuration

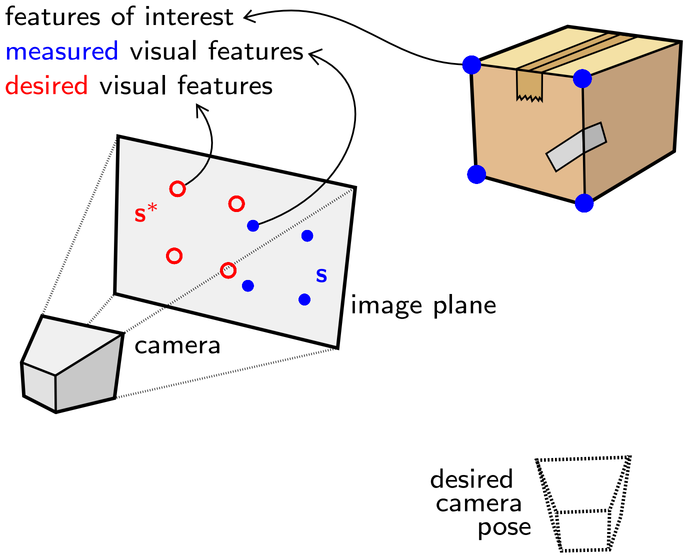




- *Eye-to-hand*: actuated target observed by a camera (left)
- *Eye-in-hand*: actuated camera observing a target (right)

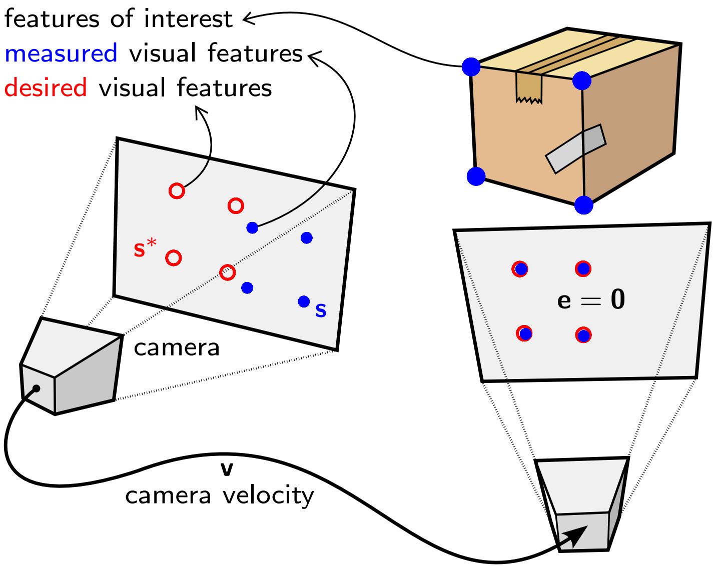
In this lecture we focus on *eye-in-hand* configurations

Working principle (with an hand-held camera)



The high-level task consists in moving the camera to a desired pose

Working principle (with an hand-held camera)



The cartesian task is actually translated in a *visual task*

Computing the VS control law

The VS control law is obtained in three steps

1. *Model design*: the features motion is related to the camera motion as

$$\dot{\mathbf{s}} = \mathbf{L}\mathbf{v} \tag{1}$$

   where $\mathbf{L}$ is the *interaction matrix*

2. *Stable error dynamics*: we want $\mathbf{s} \to \mathbf{s}^*$, that is $\mathbf{e} = (\mathbf{s} - \mathbf{s}^*) \to \mathbf{0}$

$$\dot{\mathbf{e}} = \dot{\mathbf{s}} - \dot{\mathbf{s}}^* = -\lambda\mathbf{e}, \quad \lambda > 0 \tag{2}$$

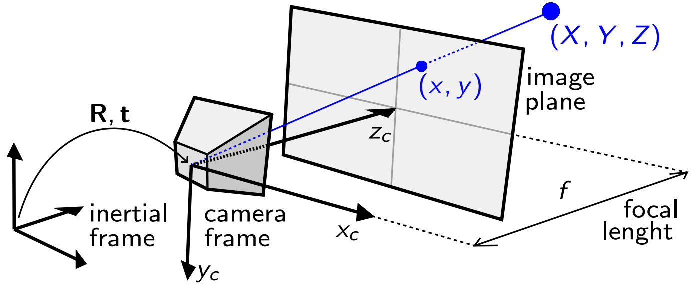
   where λ is the *control gain*

3. *Controller computation*: (1) in (2) with a constant target ($\dot{\mathbf{s}}^* = \mathbf{0}$)

$$\dot{\mathbf{e}} = -\lambda\mathbf{e} = \dot{\mathbf{s}} = \mathbf{L}\mathbf{v} \implies \boxed{\mathbf{v} = -\lambda\mathbf{L}^+\mathbf{e}}$$

Camera projection model (1/5)

- *Frontal pin-hole* camera model



- *Perspective projection*

$$\boxed{x = f\frac{X}{Z}, \quad y = f\frac{Y}{Z}}$$

- Sometimes *normalized coordinates* are used, considering $f = 1$
- Also used to computed the interaction matrix (go to slide 18)

Camera projection model (2/5)

- In a more compatc way, using *homogeneous coordinates*:

$$Z\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

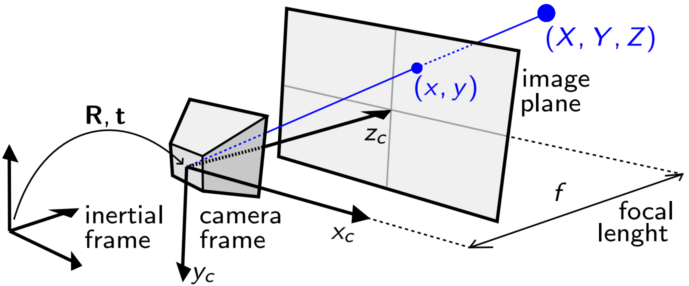
- The *depth* Z is *unkown* (remember the lost of information): we call it as *parameter* ζ in the left-hand side of the equation
- For convenience, we write the matrix as

$$\begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- In general, the Cartesian point can be expressed in the *inertial frame*

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0\ 0\ 0 & 1 \end{pmatrix}^{-1}\begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{pmatrix}$$

Camera projection model (3/5)



- The *camera ideal model* results to be

$$\zeta \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0\ 0\ 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{pmatrix}$$

Camera projection model (4/5)

- However, the features are *measured in pixels*, with coordinates (u, v), which are related to (x, y) through the following relationship

$$\boxed{u = u_0 + \frac{x}{\rho_w}, \quad v = v_0 + \frac{y}{\rho_h}}$$

where (ρ_w, ρ_h) is the *size of the pixel* and (u_0, v_0) is the *central point*

- Using homogenous coordinates and writing in compact form:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} 1/\rho_w & 0 & u_0 \\ 0 & 1/\rho_h & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- Used to compute the interaction matrix (go to slide 20)

Camera projection model (5/5)

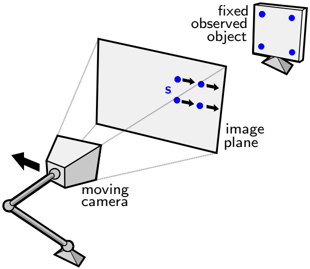
- Putting all together

$$\underbrace{\zeta \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}}_{\tilde{\mathbf{p}}} = \underbrace{\overbrace{\begin{pmatrix} f/\rho_w & 0 & u_0 \\ 0 & f/\rho_h & v_0 \\ 0 & 0 & 1 \end{pmatrix}}^{\textit{intrinsic}}}_{\mathbf{K}} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\mathbf{P}} \underbrace{\overbrace{\begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0\ 0\ 0 & 1 \end{pmatrix}^{-1}}^{\textit{extrinsic}}}_{{}^{0}\mathbf{T}_c^{-1}} \underbrace{\begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{pmatrix}}_{\tilde{\mathbf{P}}}$$

$$\boxed{\tilde{\mathbf{p}} = \underbrace{\mathbf{K}\,\mathbf{P}\,\left({}^{0}\mathbf{T}_c\right)^{-1}}_{\mathbf{C}}\,\tilde{\mathbf{P}}}$$

- $\mathbf{K}$ is called *intrinsic parameter matrix* or *calibration matrix*
- $\mathbf{P}$ is called *standard projection matrix*
- ${}^{0}\mathbf{T}_c$ is obtained with a *extrinsic calibration*
- $\mathbf{C}$ is called *camera matrix*
- f/ρ_w and f/ρ_h are the *focal lenght expressed in units of pixels*
- From $\tilde{\mathbf{p}}$ we obtain the model in pixels of our visual feature: $\mathbf{s} = (u, v)^\top$

Computation of the interaction matrix (1/3)



- The interaction matrix relates the *velocity of the feature* to the *velocity of the camera*

$$\boxed{\dot{\mathbf{s}} = \begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} = \mathbf{L}\mathbf{v} = \mathbf{L} \begin{pmatrix} \boldsymbol{\nu} \\ \boldsymbol{\omega} \end{pmatrix}}$$

- Remember: eye-in-hand configuration

- From the perspective equation (see slide 13) we have

$$\dot{x} = f\,\frac{\dot{X}Z - X\dot{Z}}{Z^2} = \frac{f}{Z}\dot{X} - \frac{x}{Z}\dot{Z}, \quad \dot{y} = f\,\frac{\dot{Y}Z - Y\dot{Z}}{Z^2} = \frac{f}{Z}\dot{Y} - \frac{y}{Z}\dot{Z}$$

- In compact form:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} \frac{f}{Z} & 0 & -\frac{x}{Z} \\ 0 & \frac{f}{Z} & -\frac{y}{Z} \end{pmatrix} \begin{pmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{pmatrix}$$

Computation of the interaction matrix (2/3)

- The time derivative of the point expressed in Camera frame is related to the velocity of the camera:

$$\begin{pmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{pmatrix} = -\boldsymbol{\nu} - \boldsymbol{\omega} \times \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

  that is

$$\begin{pmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 & 0 & -Z & Y \\ 0 & -1 & 0 & Z & 0 & -X \\ 0 & 0 & -1 & -Y & X & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{\nu} \\ \boldsymbol{\omega} \end{pmatrix}$$

- Substituting:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} -\frac{f}{Z} & 0 & \frac{x}{Z} & \frac{xY}{Z} & -f - \frac{xX}{Z} & \frac{fY}{Z} \\ 0 & -\frac{f}{Z} & \frac{y}{Z} & f + \frac{yY}{Z} & -\frac{yX}{Z} & -\frac{fX}{Z} \end{pmatrix} \begin{pmatrix} \boldsymbol{\nu} \\ \boldsymbol{\omega} \end{pmatrix}$$

Computation of the interaction matrix (3/3)

- Considering that $X = xZ/f$ and $Y = yZ/f$:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} -\frac{f}{Z} & 0 & \frac{x}{Z} & \frac{xy}{f} & -f - \frac{x^2}{f} & y \\ 0 & -\frac{f}{Z} & \frac{y}{Z} & f + \frac{y^2}{f} & -\frac{xy}{f} & -x \end{pmatrix} \begin{pmatrix} \boldsymbol{\nu} \\ \boldsymbol{\omega} \end{pmatrix}$$

- From the metric-pixel conversion (see slide 16) we have that

$$\dot{u} = \dot{x}/\rho_w, \quad \dot{v} = \dot{y}/\rho_h$$

$$x = (u - u_0)\rho_w = \bar{u}\rho_w, \quad y = (v - v_0)\rho_h = \bar{v}\rho_h$$

- Substituting:

$$\begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} = \underbrace{\begin{pmatrix} -\frac{f}{\rho_w Z} & 0 & \frac{\bar{u}}{Z} & \frac{\bar{u}\bar{v}\rho_h}{f} & -f - \frac{\bar{u}^2\rho_w}{f} & \bar{v} \\ 0 & -\frac{f}{\rho_h Z} & \frac{\bar{v}}{Z} & f + \frac{\bar{v}^2\rho_h}{f} & -\frac{\bar{u}\bar{v}\rho_h}{f} & -\bar{u} \end{pmatrix}}_{\mathbf{L}} \begin{pmatrix} \boldsymbol{\nu} \\ \boldsymbol{\omega} \end{pmatrix}$$

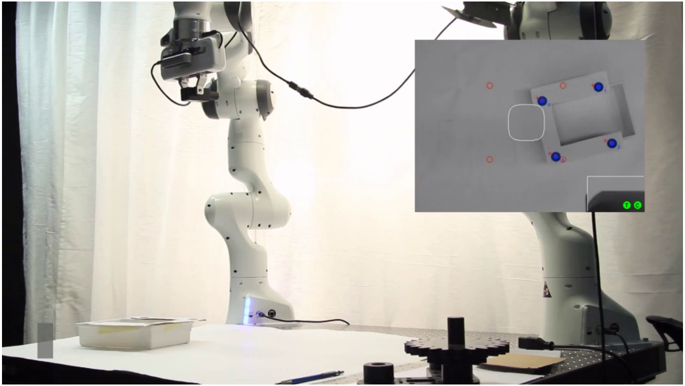
(Some) practical aspects of VS

- One point is not enough to uniquely determine the pose of the camera at convergence
- For example, if we want to control the motion of the camera in the 3D space, at least three points have to be used
- This means that the information used in the control law is the stack of three sets:

$$\mathbf{s} = \begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \mathbf{s}_3 \end{bmatrix}, \quad \mathbf{s}^* = \begin{bmatrix} \mathbf{s}_1^* \\ \mathbf{s}_2^* \\ \mathbf{s}_3^* \end{bmatrix}, \quad \mathbf{L} = \begin{bmatrix} \mathbf{L}_1 \\ \mathbf{L}_2 \\ \mathbf{L}_3 \end{bmatrix}$$

- The choise of the visual features, their number, and their desired value is part of the algorithm design

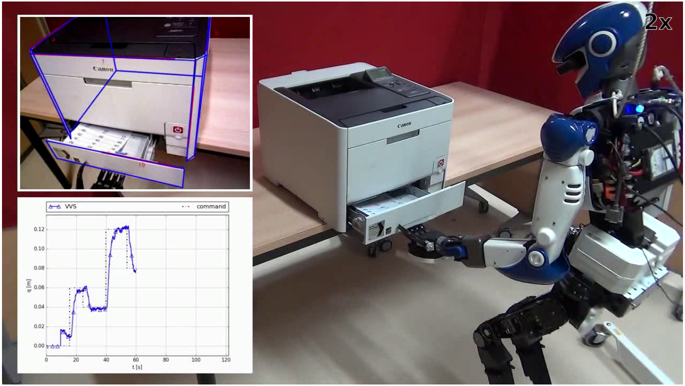
Application example (1/5): pick-and-place



[video]

Task: place an object in a box

Application example (2/5): robotic manipulation



[video]

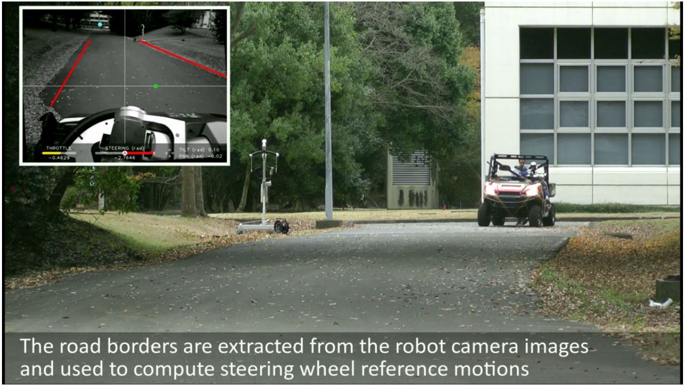
Task: open/close a drawer

Application example (3/5): corridor navigation

[video]

Task: navigate at the center of a corridor

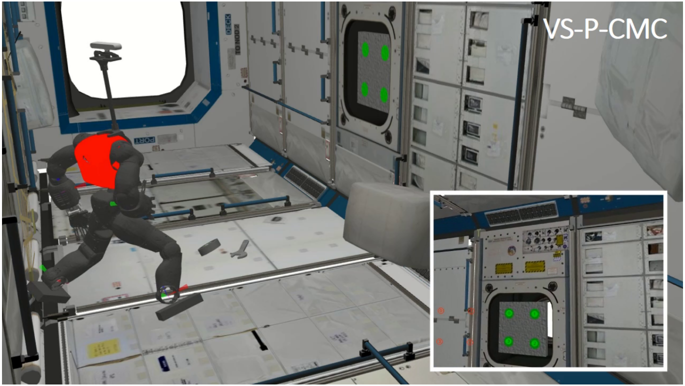
Application example (4/5): driving a car with a humanoid



[video]

Task: drive the car at the center of the road

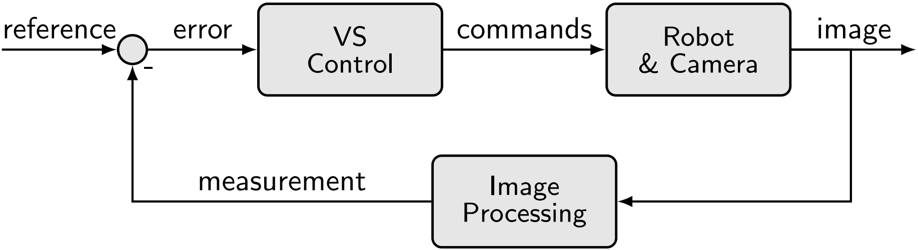
Application example (5/5): space operation with a humanoid



[video]

Task: re-orient the body with respect to a tool, in space

PBVS block diagram

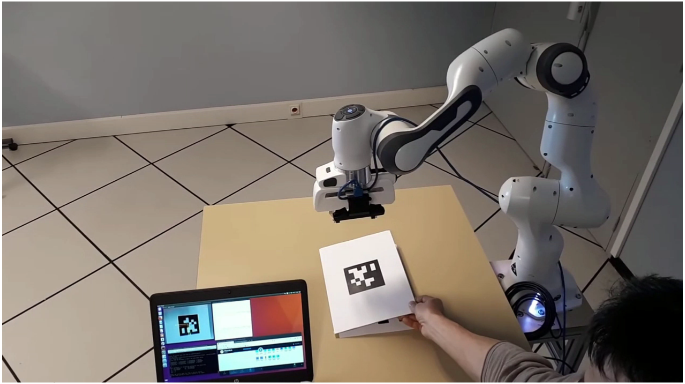


reference	desired camera pose	$\mathbf{s}^*$
measurement	current camera pose	$\mathbf{s}$
error	Cartesian error	$\mathbf{e} = \mathbf{s} - \mathbf{s}^*$
commands	velocities command	$\mathbf{v}$

PBVS and the camera pose reconstruction problem

- PBVS implies the reconstruction of the camera pose, which is normally a complex task
- A number of modules can be used
  - Model-based pose reconstruction modules
  - Fiducial marker detectors (e.g., April tag)
  - Visual Odometry
  - Visual Simultaneous Localization and Mapping (V-SLAM)
  - Machine learning-based approaches, such as self-supervised learning

A simple PBVS application example



[video]

Task: keep the robot camera at a given pose from the marker

Credits: ViSP (by INRIA Rennes, France)
`https://visp-doc.inria.fr/doxygen/visp-daily/index.html`

Final remark

- What is visual servoing?
  - Vision-based control of robot
- Why do we need visual servoing?
  - Translate cartesian tasks into visual tasks
- How do we build visual servoing?
  - Control law and visual feedback definition
- What can we do with visual servoing?
  - Navigation, manipulation, operation...

More advanced topics

- Standards VS is purely *reactive*: its performace can be improved by using *predictive* techniques, such as model predictive control
- The measurement of the visual features can be robustified by extending both control and perception algorithm; for example
  - on the control side, adaptive or weighing mechanisms can be used
  - on the perception side, machine learning tools can be employed
- VS can be applied to many different robotic platforms; for humanoids has to be computed accordingly to the whole-body motion
- Further developments deal with the integration of VS with optimization, planning and machine learning methodologies

(Some) References

1. F. Chaumette, and S. Hutchinson, "Visual servo control: Part I. Basic approaches," *IEEE Robotics & Automation Magazine*, 2006, 13(4), pp. 82–90.
2. P. Corke. "Robotics, vision and control: fundamental algorithms in MATLAB". *Springer*, 2017.
3. B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo. "Robotics: modelling, planning and control." *Springer Science & Business Media*, 2010.
4. Y. Ma, S. Soatto, J. Kosecka, S. Shankar Sastry. "An invitation to 3-D vision: from images to geometric models." *Springer Science & Business Media*, 2012.
5. A. De Luca, "Visual servoing," *slides of the Robotics 2 course given at Sapienza University of Rome*, Italy, 2020.
6. E. Marchand, F. Chaumette, "Feature tracking for visual servoing purposes," *Robotics andAutonomous Systems*, 2005, 52(1), pp.53–70.
7. E. Marchand, F. Spindler, F. Chaumette. "ViSP for visual servoing: a generic software platform with a wide class of robot control skills," *IEEE Robotics & Automation Magazine*, 2005, 12(4), pp. 40–52.
8. F. Chaumette, "Image moments: a general and useful set of features for visual servoing," *IEEE. Transactions on Robotics*, 2004, 20(4), pp.713–723.
9. C. Collewet, E. Marchand, "Photometric visual servoing," *IEEE Transactions on Robotics*, 2011, 27(4), pp.828–834.
10. A. Paolillo, A. Faragasso, G. Oriolo, M. Vendittelli, "Vision-based maze navigation for humanoid robots," *Autonomous Robots*, 41(2), pp. 293–309, 2017.

Thank you for the attention!

Q/A time