Visual Servoing
TP: Free Camera 3D Pose Control

Erol Ozgur *

January 16, 2014

1 Problem

We want to move a camera from its current Cartesian pose to a desired Cartesian pose (see Fig. 1). We assume that we can measure the Cartesian pose of the camera at every instant of time using a sensor. Please do the following exercises:

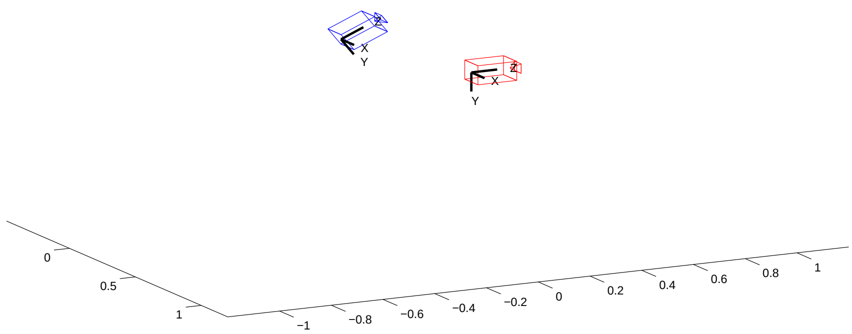

Figure 1: Camera pose control. Blue camera shows the initial pose of the camera, and the red camera shows the desired pose of the camera.

1. Discuss how to realize this 3D pose control.

2. Reformulate the problem mathematically.

3. Draw a block diagram of this 3D pose control.

*Author is with Pascal Institute, IFMA, Clermont-Ferrand, France, erol.ozgur@ifma.fr

4. Before simulation in matlab, define the steps of the algorithm for this 3D pose control.
5. Write the matlab code to simulate this 3D pose control.
6. Plot the errors of the 3D pose control versus time.

2 Problem Reformulation

Let Cartesian pose of the object be $\mathbf{X}$ and it is expressed with respect to a reference fixed frame R and in R. Let the current location be at A and the desired location be at B. Then find a control law $\xi = (\nu, \omega)$ that will update the Cartesian pose $\mathbf{X}$ of the object from $\mathbf{X}_{A/R}$ to $\mathbf{X}_{B/R}$ while time t goes from initial time t_i to final time t_f.

3 Solution

Let $s(\mathbf{X}) \in \Re^{6\times 1}$ be the another representation of the Cartesian pose $\mathbf{X} \in \Re^{4\times 4}$ as follows:

$$\mathbf{X} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}, \qquad s(\mathbf{X}) = \begin{bmatrix} \mathbf{t} \\ \mathbf{u}\,\theta \end{bmatrix} \tag{1}$$

where $\mathbf{u}\,\theta$ corresponds to the rotation $\mathbf{R}$. Here $\mathbf{u}$ is a unit rotation axis and θ is the rotation angle around this axis $\mathbf{u}$.

$$\mathbf{e} = s\left((\mathbf{X}_{B/R})^{-1} * \mathbf{X}_{A(t)/R} \right) = s\left(\mathbf{X}_{A(t)/B} \right) \tag{2}$$

Taking the time derivative of the error function when it is expressed as $\mathbf{e}$, allows us to appear the control law ξ as below:

$$\dot{\mathbf{e}} = \mathbf{L}_e\, \xi_{A(t)/B} \tag{3}$$

where $\mathbf{L}_e$ is the interaction matrix between the error dynamics and the control law. We then force the error decrease exponentially towards zero by imposing $\dot{\mathbf{e}} = -\lambda\,\mathbf{e}$. This yields the control law as follows:

$$\xi_{A(t)/B} = -\lambda\,\mathbf{L}_e^{-1}\,\mathbf{e}, \qquad \lambda > 0 \tag{4}$$

where control law $\xi_{A(t)/B}$ can be explicitly written as below:

$$\xi_{A(t)/B} = \begin{bmatrix} \nu \\ \omega \end{bmatrix}_{A(t)/B} = \begin{bmatrix} -\lambda\,\mathbf{R}^T\,\mathbf{t} \\ -\lambda\,\theta\,\mathbf{u} \end{bmatrix}_{A(t)/B} \tag{5}$$

We should then express the control law $\xi_{A(t)/B}$ with respect to the fixed reference frame R rather than desired pose frame B. This can be done as follows:

$$\xi_{A(t)/R} = \begin{bmatrix} \mathbf{R} & [\mathbf{t}]_\times\,\mathbf{R} \\ \mathbf{0}_{3\times 3} & \mathbf{R} \end{bmatrix}_{B/R} \xi_{A(t)/B} \tag{6}$$

where rotation matrix $\mathbf{R}$ and the translation vector $\mathbf{t}$ come from the Cartesian pose $\mathbf{X}_{B/R}$ of the desired location at B expressed with respect to the fixed reference frame at R. Notation $[\mathbf{t}]_\times$ implies the skew symmetric matrix of the given translation vector $\mathbf{t}$.

Afterwards, we can update the current Cartesian pose of the camera as follows:

$$\mathbf{X}_{A(t+\Delta t)/R} = \begin{bmatrix} \Delta t\,[\omega]_\times & \Delta t\,\nu \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}_{A(\Delta t)/R} \mathbf{X}_{A(t)/R} \tag{7}$$

4 Matlab Simulation

Open the file "FreeCamera3DPoseControlTP.m", which is located in the folder "tp_matlab_student", and then complete it with your servoing algorithm to simulate the 3D pose control.

Appendix - Matlab Functions

$\mathbf{x}$ = **uthetat2dq(u, θ, t)** computes the dual quaternion $\mathbf{x}$ from given rotation unit axis $\mathbf{u}$, rotation angle θ, and translation vector $\mathbf{t}$.

(u, θ, R, t) = **dualq2uthetaRt(x)** computes the rotation unit axis $\mathbf{u}$, rotation angle θ, rotation matrix $\mathbf{R}$, and translation vector $\mathbf{t}$ from given dual quaternion $\mathbf{x}$.

$\mathbf{x}$ = **muldualpq(p, q)** multiplies two given dual quaternions, e.g., $\mathbf{p}$ and $\mathbf{q}$, and gives the result dual quaternion, e.g., $\mathbf{x}$.

$\mathbf{x}^{-1}$ = **conjdualqsimple(x)** computes the inverse of a given dual quaternion.

plot_pose(X, color) plots the 3D frame of a given Cartesian pose $\mathbf{X}$ in a defined color, e.g., for red 'r', for blue 'b'.

plot_camera(X, color) plots the camera at a given Cartesian pose $\mathbf{X}$ with a defined color, e.g., for red 'r', for blue 'b'.

$\mathbf{s}$ = **skew(t)** computes the skew symmetric matrix, e.g., $\mathbf{s}$, of a given vector, e.g., $\mathbf{t}$.

There are also the following standard functions of Matlab which can help you to visualize the 3D pose control: **clf**; **hold on**; **plot3**; **drawnow**; **axis**; **view**.