Marine Mechatronics
Practical work n°4

Implementation of a visual servoing on the BlueRov

by Associate Prof. Claire Dune and Prof. Vincent Hugel (COSMER, Université de Toulon)

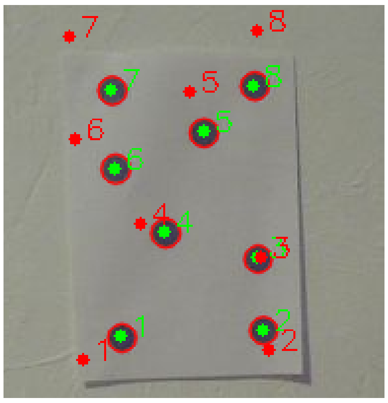
Visual servoing implementation

In this tutorial, we will experiment the visual kinematic servoing on a Blue Rov underwater robot. We will consider N points on a planar target. The objective of the visual servo control is to move the robot so that the characteristics s defined from the tracked points (green) correspond to the desired characteristics s*, calculated from the 2D points extracted at the desired position (red).

First, you will work out of the water and when your control is ready, you can test in the water. For you report, you will have to save :

- **the bags of your experiment with all the topics**
- **the graphs obtained with plot juggler or rqtplot**

Step 1 : 2D point detection and tracking



The first step is to detect information in the image. The second step is to compute features from this image information to design s and s*.

1. Follow the **READ ME** file to play a bag and test the tracker node.
2. In a terminal **use rqt_plot or plotjuggler** to display the topics relative to tracked points and desired points
3. In **visual_servoing_mir.py line 93,** you will find the tracker call back that listens to the tracked point topic. The points are converted from pixel to meter at line 106, 107**. For your information,** the camera calibration values are given lines 6 to 11 in the file camera_parameters.py. They correspond to the usb cam calibration.
4. **At line 110, you have to write the code to compute the error between the desired and current 2D point coordinates.** It is published at line 167.
5. In a terminal **use rqt_plot or plotjuggler** to display the topics relative the visual servoing error

Interaction matrix

For every feature s, the keypoint of the visual servoing control law is to find the relation between the derivative of s in time and the motion of the camera vcam = (vx,vy,vz,ωx,ωy,ωz).
It is the interaction matrix L.

$$\boxed{ds/dt = Ls\ vcam \qquad (1)}$$

For a feature s=(x,y) that is the coordinate of a 2D point, the interaction matrix is the following (you have to define an arbitrary Z, which is the depth of the point)

Ls =

$$\begin{bmatrix} -1/Z & 0 & x/Z & xy & -(1+x^2) & y \\ 0 & -1/Z & y/Z & 1+y^2 & -xy & -x \end{bmatrix}$$

In the file visual_servoing.py, line 18, you will find the implementation of L and at line 26, the implementation of L for a stack of points (x1,y1,x2,y2,x3,y3…xN,yN)^T. **In the file visual_servoing_mir.py, the matrix L is computed at line 112.**

Compute the Control Law

The control law is the following :

$$\boxed{vcam = -\lambda\ Ls^{+}\ (s-s^{*}) \qquad (2)}$$

1. **In the file visual_servoing.py, line 115,** compute the control law using the interaction matrix and the error previously defined. In python the pseudo inverse is :

```
np.linalg.pinv (L)
```

2. In a terminal **use rqt_plot or plotjuggler** to display the topics relative to visual servoing error and to the camera velocity

Do not forget that this control law gives the velocity expressed in the body frame of the camera. We then have to change its frame to find the robot velocity.
We can use the twist matrix rVc :

$$\boxed{vrobot = rVc\ vcam \qquad (3)}$$

$\mathbf{v} = (\boldsymbol{\upsilon}, \boldsymbol{\omega})$: kinematic screw between the camera and the scene expressed at $\mathbf{C}$ in $\mathcal{F}_c$

$\boldsymbol{\omega}$: rotational velocity : $\qquad [\boldsymbol{\omega}]_\times = {}^o\mathbf{R}_c^\top\, {}^o\dot{\mathbf{R}}_c = -{}^o\dot{\mathbf{R}}_c^\top\, {}^o\mathbf{R}_c$

$\boldsymbol{\upsilon}$: translational velocity at $\mathbf{C}$: $\qquad \boldsymbol{\upsilon}(\mathbf{O}) = \boldsymbol{\upsilon}(\mathbf{C}) + [\boldsymbol{\omega}]_\times\ \mathbf{CO}$

To express $\mathbf{v}$ at $\mathbf{O}$ in $\mathcal{F}_o$: ${}^o\mathbf{v} = {}^o\mathbf{V}_c\ \mathbf{v}$ with ${}^o\mathbf{V}_c = \begin{bmatrix} {}^o\mathbf{R}_c & [{}^o\mathbf{t}_c]_\times {}^o\mathbf{R}_c \\ \mathbf{0}_3 & {}^o\mathbf{R}_c \end{bmatrix}$

1. **In the file visual_servoing.py, line 127,** change the frame of expression of the velocity from camera to robot. If you need to train to find the right transform, you can use the file **testTransform.py.**

2. In a terminal **run rqt_plot or plotjuggler** to display the topics relative to visual servoing error and to the camera velocity

Compute control values for mavros

In the file visual_servoing.py, line 176-181, the velocity are translated to mavros values.
You have to fix the sign of the parameter to send. It depends on your robot.
Take your time to test before launching the control.

```
roll_left_right = mapValueScalSat(vel.angular.x)
yaw_left_right = mapValueScalSat(-vel.angular.z)
ascend_descend = mapValueScalSat(vel.linear.z)
forward_reverse = mapValueScalSat(vel.linear.x)
lateral_left_right = mapValueScalSat(-vel.linear.y)
pitch_left_right = mapValueScalSat(vel.angular.y)
```

Test the control in air

Once you have proved your robot values to be coherent with the error, you can armed the robot and click on the A button then check the thrusters.

STOP UNTIL THE PROFESSOR HAS CHECKED

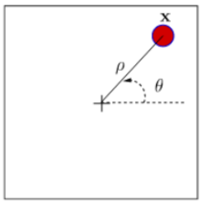
Launch the control in the water

Your ready to go into the water to test your 6 dof visual servoing.

Change feature set, interaction matrix and control behavior

Use of (ρ, θ) for an image points instead of (x, y):

$$\rho = \sqrt{x^2 + y^2}\ ,\ \theta = \arctan \frac{y}{x}$$



Corresponding interaction matrix:

$$\mathbf{L}_\rho = [\ \frac{-\cos\theta}{Z}\ \ \frac{-\sin\theta}{Z}\ \ \frac{\rho}{Z}\ \ (1+\rho^2)\sin\theta\ \ -(1+\rho^2)\cos\theta\ \ \ 0\ ]$$

$$\mathbf{L}_\theta = [\ \frac{\sin\theta}{\rho Z}\ \ \frac{-\cos\theta}{\rho Z}\ \ 0\ \ \frac{\cos\theta}{\rho}\ \ \frac{\sin\theta}{\rho}\ \ -1\ ]$$

1. **In the file visual_servoing.py,** take example of the matrices for 2D coordinates to build a function for a point in a rho, theta representation**.**
2. **In the file visual_servoing.py,** write the function to stack matrices for several rho theta point**.**
3. **In the file visual_servoing_mir,** change the vector of points for a vector of rho and theta representation
4. **In the file visual_servoing_mir** compute the associated error
5. Launch the control law and compare the behavior with previous control
6. Look in the state of the art to do the same for a control on other features.