ESIR3-IN : (IMED2) TP A.Krupa Medical robotics: ultrasound dense visual servoing

alexandre.krupa@inria.fr

1 Objectives

The objective of this course is to simulate the automatic positioning of an ultrasound 2D probe actuated by a medical robot. This automatic positioning will be performed thanks to a visual servoing approach that uses directly dense visual information observed in the ultrasound 2D image.

The robotic task will consist in automatically synchronizing the ultrasound probe motion with soft tissue motion in such a way to compensate the patient physiological motion (breathing). This task can be activated for example during an echographic examination in order to actively compensate the patient motion and therefore to provide to the clinician a stable ultrasound image of a moving organ of interest.

To this end you will use the UsSimulator library developed at Inria in the Rainbow team that simulates a virtual 2D ultrasound probe actuated by a virtual robot that has 6 degrees of freedom (DOF). You will also use the ViSP library developed at Inria to implement the robotic control law. ViSP documentation is available at http://visp.inria.fr

In order to install all the software required for this TP (including also an automatic installation of ViSP) follow these instructions:

This installation will compile all the required libraries. Its is very slow = 20-25 minutes.

- First verify that you have a least 300 Mo free on your linux account
- Create a temporary directory `tp4_install` in the `/tmp/` of your linux machine
```
cd /tmp
mkdir tp4_install
```
- Download with your internet browser and put in `/tmp/tp4_install` the file archive located at (available only until December 7)
https://filesender.renater.fr/?s=download&token=eb3c9cf3-b2de-4c35-862f-c7d3c728c296
- Decompress the archive
```
cd /tmp/tp4_install
tar -xvf script_install_tp4_ESIR.tar
```
- Execute the installation script:
```
cd /tmp/tp4_install/script_install_tp4_ESIR
```

```
sh script
```

The automatic installation and compilation process of the required libraries will take 20-25 minutes. During this time you can read the subject of the TP.

TEST the codes:

- When the installation is finished go in the tp4 code directory that was installed.
  ```
  cd ~/tp4_ESIR/tp4
  ```
- Then launch Eclipse C/C++ Debugger application (it is an C++ editor for coding and compiling). Select the default proposed workspace directory `/homedir/your_username/workspace` and validate it by pressing OK. Then select "Import existing projects" and in "Select root directory" browse to the path `tp4_ESIR/tp4/tp4-ultrasound-dense-vs`. In the next window press the "Finish" button.
  If you are not familiar with Eclipse then ask me to come to show you how to compile (build) the project codes.
- When the project is built, open a terminal and type the following command to set the path of shared libraries required for execution (do be done each time you open a new terminal):
  ```
  export LD_LIBRARY_PATH=$HOME/tp4_ESIR/third-party/VTK-9.3.0-build/lib:$LD_LIBRARY_PATH
  ```

  Then go at the location of the application binary file by:
  ```
  cd ~/tp4_ESIR/tp4/tp4-ultrasound-dense-vs/tp/
  ```

  and run and test the application binary outside Eclipse by the command:
  ```
  ./servo-simu-dense
  ```

If you need to complete the TP at home, you will find a virtual machine with Ubuntu 14.04.2 LTS 64 bit. A user with login `lagadic` and password `lagadic` was created. All the softwares required in this course are already installed on the virtual machine. The image of this virtual machine can be downloaded at (available only until December 7)
https://filesender.renater.fr/?s=download&token=1908f542-2c5e-4dc2-b5b1-f14e0b977232
Download and install the VirtualBox client https://www.virtualbox.org to load this virtual machine.

You will write your answers and comments to the questions in a report (pdf format using latex or Microsoft word). In this pdf you will also put the missing codes you need to complete in the TP.
Send me by email to `alexandre.krupa@inria.fr` **your report before December 6** (so you have one week to finish the TP at home) in a **zip archive that also contains the .cpp files**. **The report will be graded (evaluation).**
To insert your codes in the latex document use the latex package `\usepackage{verbatim}` and put (copy/paste) your code like this:

```
\begin{verbatim}
your c++ code
\end{verbatim}
```

2 Ultrasound simulator: UsSimulator

The UsSimulator library provides a ultrasound image simulator. It allows the positioning of a virtual 2D ultrasound probe in a real ultrasound volume that was previously captured. The main functionalities of this simulator are:

- To load a 3D ultrasound volume composed of a set of parallel ultrasound slices that were previously captured at constant interval distances.
- To position the virtual probe at a given pose with respect to a Cartesian world frame.
- To provide the current pose of the probe with respect to the world frame.
- To apply displacement velocities (3 translational velocities, 3 rotational velocities) to the frame attached to the virtual probe.
- To apply displacement velocities (3 translational velocities, 3 rotational velocities) to the frame attached to the volume in order to simulate rigid motion of soft tissue.
- To generate and display the ultrasound image (grey-level) observed by the virtual probe thanks to interpolation technique.
- To visualize in the 3D scene, the slice observed by the virtual probe and 3 orthogonal slices centered in the volume.

3 Ultrasound dense visual servoing

We recall hereafter the principle of the ultrasound dense visual servoing approach. The visual information (visual features vector) used in the control scheme, denoted $\mathbf{s}$, contains directly the intensities of the pixels of a region of interest (ROI) such that:

$$\mathbf{s} = (I_{1,1}, ..., I_{u,v}, ..., I_{M,N}) \tag{1}$$

where M, N are respectively the width and the height of the ROI, $I_{u,v}$ is the intensity, expressed in grey level and (u, v) are the 2D pixel coordinates in the US image.

The interaction matrix $L_{I_{u,v}}$ that relates the variation of the intensity of a pixel point to the probe velocity $\mathbf{v}_p$ such that $\dot{I}_{u,v} = L_{I_{u,v}} \mathbf{v}_p$, is given by:

$$L_{I_{u,v}} = \left[\begin{array}{cccccc} \nabla I_x & \nabla I_y & \nabla I_z & y\nabla I_z & -x\nabla I_z & x\nabla I_y - y\nabla I_x \end{array} \right] \tag{2}$$

in which the meter coordinates of the pixel point are deduced from the coordinates of the pixel (u, v) in the image and from the intrinsic parameters of the probe:

$$\left(\begin{array}{c} x \\ y \end{array} \right) = \left(\begin{array}{c} s_x(u - u_0) \\ s_y(v - v_0) \end{array} \right) \tag{3}$$

with (s_x, s_y) the size of one pixel and (u_0, v_0) the pixel coordinates of the center of the ultrasound 2D image. Here $\nabla I_{u,v} = [\nabla I_x \;\; \nabla I_y \;\; \nabla I_z]$ corresponds to the 3D image gradient associated with the pixel (u, v). It comprised three components, $\nabla I_x = \dfrac{\partial I_{u,v}}{\partial x}$, $\nabla I_y = \dfrac{\partial I_{u,v}}{\partial y}$ and $\nabla I_z = \dfrac{\partial I_{u,v}}{\partial z}$ that describe the intensity variation of the pixel (u, v) along the three orthogonal axes of the US image frame. These three components are obtained with three 3D derivative filters applied to a thin volume composed of 3 parallel slices that is centered on the current image slice.

The complete interaction matrix $\mathbf{L}_s$, that relates the variation of the visual features to the probe velocity $\mathbf{v}_p$ such that $\dot{\mathbf{s}} = \mathbf{L}_s \mathbf{v}_p$, is then built by stacking the $M \times N$ matrices $\mathbf{L}_{I_{u,v}}$:

$$\mathbf{L_s} = \begin{bmatrix} L_{I_{1,1}} \\ \vdots \\ L_{I_{M,N}} \end{bmatrix} \quad (4)$$

Finally, the control velocity $\mathbf{v}_p$ applied to the virtual US probe is computed so as to minimize the visual error $\mathbf{e}(t) = \mathbf{s}(t) - \mathbf{s}^*$ where $\mathbf{s}^*$ corresponds to the vector containing the desired pixel intensities of the ROI. To obtain an exponential decrease of the visual error, the following classical control law is used:

$$\mathbf{v}_p \;=\; -\lambda\, \widehat{\mathbf{L_s}}^{+} \left(\mathbf{s}(t) - \mathbf{s}^*\right) \quad (5)$$

where λ is the controller gain and $\widehat{\mathbf{L_s}}^{+}$ is the Moore-Penrose pseudo-inverse of an estimation $\widehat{\mathbf{L_s}}$ of the interaction matrix $\mathbf{L_s}$.

4 Implementation of the ultrasound dense visual servoing

In this part you will code the control law of the ultrasound dense visual servoing approach presented in the previous section.
First analyse the code files in the sub-directory `tp` of the project. As you can see several parts located after the comments "Question x" are missing. Compile (build) the project and see what is happening when you run it in a terminal (outside Eclipse) by the command `./servo-simu-dense`. As you notice, for the moment no visual servoing is performed due to the missing codes.

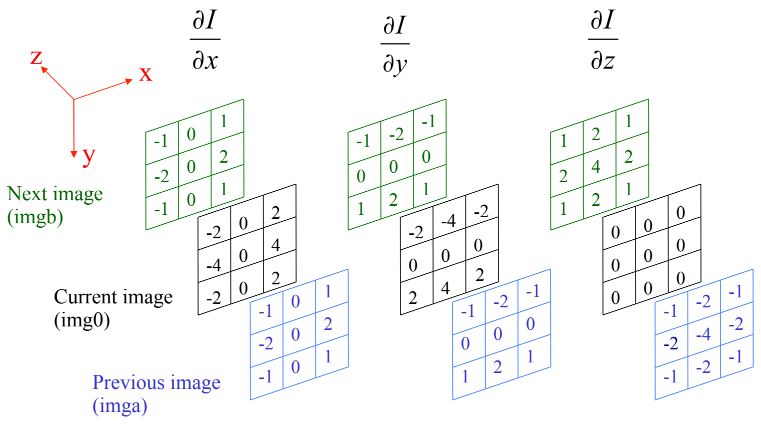
To complete the code, follow the following instructions:

Question 1 *Complete the code in the while(1) main loop of* `servo-simu-dense.cpp` *to apply a 6-DOF sinusoidal velocity to the ultrasound volume (object) in order to simulate the physiological motion of the patient. The period of the sinusoidal signals has to be set to 5 seconds and the amplitude to 0.005 meters/s for the translational components and 5 deg/s for the rotational components. Compile and execute. What do you observe? Comments the result (current ultrasound image, difference image and probe pose error curves).*

Question 2 *Now compute the desired visual features vector (equation (1)) containing the intensities of the pixels inside the ROI at the initial = desired probe location.*

Question 3 *To determine the interaction matrix (equation (2)) related to the intensity of a pixel, you need first to compute the 3D gradient $\nabla I_{u,v}$. To this end, implement in* `usImageGradient.cpp` *the 3D spatial derivative filters presented in figure 1.*

Question 4 *In* `usIntensityInteraction.cpp` *Implement the code that computes the interaction matrix $\mathbf{L}_s$ related to the visual features vector using equations (2), (3) and (4). Note that in case of a motion compensation task we will use the constant interaction matrix computed at the initial pose of the probe that also corresponds to the one obtained at convergence of the visual servoing.*



$\frac{\partial I}{\partial x}$
$\frac{\partial I}{\partial y}$
$\frac{\partial I}{\partial z}$
z
x
y
Next image (imgb)
Current image (img0)
Previous image (imga)

Figure 1: 3x3 spatial derivative filters

Question 5 *Complete the code in the while(1) main loop of* `servo-simu-dense.cpp` *to update the current visual features vector with the pixel intensities of the ROI of the current ultrasound image.*

Question 6 *Code the control law (equation (5)) that provides the velocity to apply to the probe by setting a control gain value of 0.4 and using the pseudoInverse() method of ViSP.*

Question 7 *As the dimension of the visual features error corresponds to the number $nbPx$ of pixels inside the ROI, it is not easy to visualize all its components on a same plot. Therefore we will plot a normalize visual cost function instead that we defined by the Euclidean norm $\|(\mathbf{s} - \mathbf{s}^*)\|/nbPx$. Implement the code of this visual cost function.*

5 Interpretation of the results

Question 8 *Execute the code and comment the results observed in the 3D view, ultrasound image, difference image and curves.*

Question 9 *Test the simulation with different sizes of the ROI from huge size (close to the whole ultrasound image) to very tiny ROI (close to 3x3 pixels size). What do you observe? Comment and justify the reason of this behaviour.*

Question 10 *Define a correct ROI and test different values of the control gain. What do you observe when the control gain is divided by 4 ($\lambda = 0.1$)? and when the control gain is multiplied by 2 ($\lambda = 0.8$)? What is the maximum control gain value that guaranties a correct behaviour of the motion compensation task?*

Question 11 *Multiply the amplitude of the sinusoidal velocities applied to the ultrasound volume (object) by 4 in order to simulate fast physiological motion of the patient. What do you observe? How does behave the curve of the pose error and visual cost function? Is it possible to tune the control gain in order to efficiently compensate the motion without obtaining instability on the probe motion? try and comment.*

Question 12 *In order to compensate the tracking error one solution consists in adding an integral* *term in the control law such that the new control law becomes:*

$$\mathbf{v}_p \quad = \quad -\lambda\, \mathbf{L_s}^{+}\, (\mathbf{s}(t) - \mathbf{s}^{*}) - \mu \mathbf{L_s}^{+} \textstyle\sum_j \mathbf{e}(j) \tag{6}$$

with μ the integral gain and $\mathbf{e}(j)$ the sum of visual features errors over the time. Add this integral term in the control law with $\mu = 0.05$ and test the behaviour of the visual servoing for the same velocity amplitude applied to the volume than in the previous question. You will first set the control gain to its default value $\lambda = 0.4$. What to you observe? Comment the results and different curves. Then test different values of the integral term and comment on the results. Modify the control gain to $\lambda = 0.1$, and comment on the results obtained for different integral term values. What rule on the adjustment between λ and μ allows a good behaviour of the system?

Question 13 *Now we will test the ultrasound dense visual servoing approach for an automatic positioning task that consists in reaching a desired image that is different from the initial one. To this end, first deactivate the sinusoidal motion applied to the volume by putting in comments your code of Question 1 and then set the initial probe pose (Question 13 in the code) to a slightly different pose that the desired one by applying to the probe cartesian frame relative displacements of 0.002 meters along each translation axis, 2 deg around its x and y axes and 4 deg around its z axis. Comment the obtained results. Now increase these relative initial displacements and launch again the simulation. What do you observe? Is this approach more suitable for a probe positioning task or for a motion compensation task?*