University of Ljubljana, Faculty of Electrical Engineering
Robot Vision (RV)

April 19/20, 2016

Exercise 6: Spatial Transformation and Registration

Created by: Žiga Špiclin | http://lit.fe.uni-lj.si/RV | Homework deadline: May 3/4, 2016

Instructions

Spatial or geometric transformations $\mathcal{T}: \mathbb{R}^2 \to \mathbb{R}^2$ or $\mathcal{T}: \mathbb{R}^3 \to \mathbb{R}^3$ are used to map pixel positions (x, y) or (x, y, z) of 2D or 3D images, respectively, whereas the pixel intensity values are not changed. In this way we can enlarge or shrink an image through scaling, translate or rotate an image, or apply other linear and nonlinear geometric transformations to the image. An arbitrary geometric transformation of 2D or 3D image can be written as:

$$(u, v) = \mathcal{T}(x, y) \quad \text{oz.} \quad (u, v, w) = \mathcal{T}(x, y, z)\,.$$

Most general linear geometric transformation is the **affine transformation**, which is composed of scaling, translation, rotation and skew. In 2D the affine transformation is determined by 6, while in 3D it is determined by 12 parameters:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathcal{T}_{affine}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\mathcal{T}_{affine}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

where parameters t_x, t_y in t_z denote translations along the x, y and z image axes and parameters a_{ij} encode scaling, rotation and skew. Matrix of the affine transformation can be composed by sequential multiplication of homogeneous matrices of the elementary transformations, where the order of multiplication from right to left represents the order of applied transformations, for instance:

$$\mathcal{T}_{affine} = \mathcal{T}_{trans}\, \mathcal{T}_{skew}\, \mathcal{T}_{rot}\, \mathcal{T}_{scale}\,.$$

Rigid transformation, which is obtained by composing the elementary rotation and translation matrices and ki jo dobimo z združevanjem rotacijske in translacijske elementarne preslikave in **similarity transformation**, which is obtained by composing the rigid transformation and scaling, are often used in practical applications.

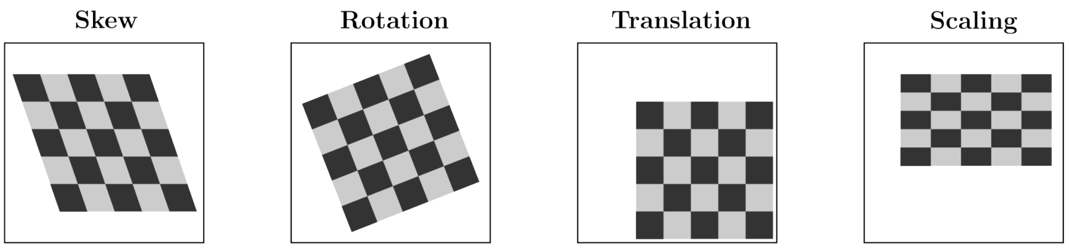
Geometric registration of two shapes is a process of finding an optimal geometric transformation $\mathcal{T}$, which aligns common structure or features of the two shapes. Shapes are often represented as point clouds, thus a simple similarity measure between two shapes is the mean square Euclidean distance, computed for corresponding points on the two shapes. Given a set of K corresponding points, the transformation $\mathcal{T}$ can be computed such that the transformed points $\mathcal{T}(x_i, y_i)$ of a reference shape best align with the points of the input shape (u_i, v_i), or vice versa E.g.:

$$(u_i,\, v_)) \leftrightarrow \mathcal{T}(x_i,\, y_i) \quad \text{or} \quad (x_i,\, y_i) \leftrightarrow \mathcal{T}^{-1}(u_i,\, v_i)\,.$$

Based on the number of corresponding points and the type of transformation, the points can be either exactly aligned through **interpolative registration**, i.e. $\mathcal{T}(x_i,\, y_i) = (u_i,\, v_i)$, or approximately aligned through **approximative registration**, i.e. $\mathcal{T}(x_i,\, y_i) \approx (u_i,\, v_i)$.

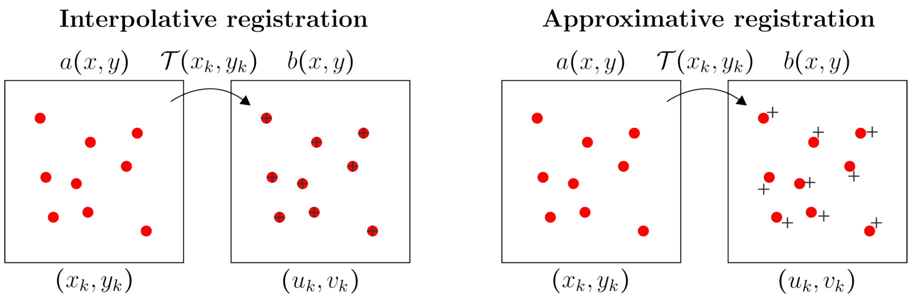
Affine interpolative registration between two 2D shapes is uniquely determined given three pairs $(K = 3)$ of noncolinear corresponding points $(u_i,\, v_i) \leftrightarrow (x_i, y_i)$, while registration of two 3D shapes requires six pairs $(K = 6)$. In 2D the affine interpolative registration $\mathcal{T}$ between three pairs of points is given as:

$$\mathcal{T} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix}^{-1}\,.$$



$$\mathcal{T}_{2D}=\begin{bmatrix} 1 & g_{xy} & 0 \\ g_{yx} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathcal{T}_{2D}=\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathcal{T}_{2D}=\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad \mathcal{T}_{2D}=\begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathcal{T}_{3D}=\begin{bmatrix} 1 & g_{xy} & g_{xz} & 0 \\ g_{yx} & 1 & g_{yz} & 0 \\ g_{zx} & g_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathcal{T}_{3D}=\mathcal{T}_\alpha\,\mathcal{T}_\beta\,\mathcal{T}_\gamma \quad \mathcal{T}_{3D}=\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathcal{T}_{3D}=\begin{bmatrix} k_x & 0 & 0 & 0 \\ 0 & k_y & 0 & 0 \\ 0 & 0 & k_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Affine approximative registration between two 2D shapes can be determined, when given more than three pairs ($K > 3$) of corresponding noncolinear points. In this way we have to solve an overdetermined system of equations, for instance by minimizing the mean square Euclidean distance between the corresponding points:

$$\Sigma^2 = \frac{1}{K}\sum_{i=1}^{K} \|(u_i,\, v_i) - \mathcal{T}(x_i,\, y_i)\|^2 \,.$$

Closed-form solution for the unknown transformation parameter values can be obtained by deriving the above expression with respect to the unknown parameter and then equate the computed derivative expression to zero. For a 2D affine approximative registration, this process yield six equations for each of the six parameters that altogether form the following system of equations:

$$\underbrace{\begin{bmatrix} \overline{xx} & \overline{xy} & \overline{x} & 0 & 0 & 0 \\ \overline{xy} & \overline{yy} & \overline{y} & 0 & 0 & 0 \\ \overline{x} & \overline{y} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \overline{xx} & \overline{xy} & \overline{x} \\ 0 & 0 & 0 & \overline{xy} & \overline{yy} & \overline{y} \\ 0 & 0 & 0 & \overline{x} & \overline{y} & 1 \end{bmatrix}}_{\mathcal{P}_{xy}} \underbrace{\begin{bmatrix} a_{11} \\ a_{12} \\ t_x \\ a_{21} \\ a_{22} \\ t_y \end{bmatrix}}_{\mathbf{t}} = \underbrace{\begin{bmatrix} \overline{ux} \\ \overline{uy} \\ \overline{u} \\ \overline{vx} \\ \overline{vy} \\ \overline{v} \end{bmatrix}}_{\mathbf{p}_{uv}} ,$$

where the overlined elements denote the mean values, e.g. $\overline{vx} = \frac{1}{K}\sum_{i=1}^{K} u_i x_i$ and $\overline{x} = \frac{1}{K}\sum_{i=1}^{K} x_i$. The above linear system of equations can be written in matrix form as $\mathcal{P}_{xy}\,\mathbf{t} = \mathbf{p}_{uv}$, where vector $\mathbf{t}$ encodes the six parameters of the 2D affine transformation $\mathcal{T}$. Parameters of the affine registration are obtained by solving the linear system as $\mathbf{t} = \mathcal{P}_{xy}^{-1}\,\mathbf{p}_{uv}$.

Registration of two shapes is difficult because the corresponding points are generally not known in advance and need to be determined as part of the registration. A state-of-the-art method that determines the point correspondence is the so-called **iterative closest point (ICP)**. In each step k of the ICP method the corresponding point pairs are determined as the closest points $(u_i,\, v_i) \leftrightarrow (x_i,\, y_i)$ of the two shapes, whereas each point $(u_i,\, v_i)$ may correspond to only one point $(x_i,\, y_i)$. Then an interpolative or approximative registration between the corresponding points is computed to align the shapes. The moving shape is transformed by $\mathcal{T}_k$ and the above described process is repeated in the next step.

The ICP steps are repeated until transformation is nearly identity $\|\mathcal{T}_k - I\|_\infty > \varepsilon$, where ε is a small constant, or until a maximum number of iterations k_{max} is reached. If the transformations $\mathcal{T}_k$ in each step are linear, then the overall transformation after the k-th step is obtained by sequential multiplication of the partial transformations obtained in each step, for instance:

$$\mathcal{T} = \mathcal{T}_k \, \mathcal{T}_{k-1} \, \mathcal{T}_{k-2} \cdots \mathcal{T}_1 \, .$$

During this exercise you will use a collection of 2D shape given in file `letala.npy`, which can be loaded into Spyder–Python environment using `numpy.load()` and which contains a `list` variable of 30 different shapes. Each shape are described by a `numpy` array of (x, y) coordinates. During the lab work you will write functions for 2D affine transformation, affine interpolative and approximative registration and iterative closest point method.

1. Write a function that creates a 3×3 matrix of an arbitrary 2D affine transformation:

   ```
   def transAffine2D( iScale, iTrans, iRot, iShear ):
           return oMat2D
   ```

   where `iScale` is a two-row vector with scale parameters k_x, k_y, `iTrans` a two-row vector with translation parameters t_x, t_y, `iRot` an angle of rotation α in degrees and `iShear` a two-row vector of skew parameters g_x, g_y. Function should return a homogeneous 3×3 matrix `oMat2D`. Verify the function by transforming a shape independently applying arbitrary elementary transformations like skew, rotation, translation and scaling and then jointly the complete 2D affine transformation. Note that the points should be given in homogeneous coordinates. Display the original and transformed points using `matplotlib.pylab.plot()`.

2. Write a function that computes a 2D affine intepolative registration between corresponding points:

   ```
   def mapAffineInterp2D( iPtsRef, iPtsMov ):
           return oMat2D
   ```

   where `iPtsRef` is an array of coordinates of reference points, `iPtsMov` an array of moving points. The dimensions of both arrays is 3×2. In the function, the matrix inverse can be computed either analitically or using `numpy.linalg.inv()`. Function should return a 2D affine transformation in the form of homogeneous 3×3 matrix `oMat2D`. Verify the function by transforming **three points** of a shape using an arbitrary 2D affine transformation and the compute the 2D affine intepolative registration. The obtained transformation in `oMat2D` should be equal to the 2D affine transformation used to transform the points.

3. Write a function that computes a 2D affine approximative registration between corresponding points:

   ```
   def mapAffineApprox2D( iPtsRef, iPtsMov ):
           return oMat2D
   ```

   where `iPtsRef` is an array of coordinates of reference points, `iPtsMov` an array of moving points. The dimensions of both arrays is $K \times 2$, where $K \geq 3$. Function should return a 2D affine transformation in the form of homogeneous 3×3 matrix `oMat2D`. Verify the function by transforming **all points** of a shape using an arbitrary 2D affine transformation and the compute the 2D affine intepolative registration. The obtained transformation in `oMat2D` should be equal to the 2D affine transformation used to transform the points.

4. Write a function for registration of two shapes by the iterative closes point method (ICP):

   ```
   def alignICP( iPtsRef, iPtsMov, iEps, iMaxIter ):
           return oMat, oErr
   ```

   where `iPtsRef` and `iPtsMov` are arrays of coordinates of reference and moving points, respectively. The dimensions of both arrays is $K \times 2$. Parameters `iEps` and `iMaxIter` determine the stopping criteria for ICP, where `iEps` is the tolerance on the smallest change of the transformation, while `iMaxIter` is the maximum number of iterations k_{max}. Use a 2D affine approximate to register corresponding points in each step of the ICP. Function should return a 2D affine transformation in the form of homogeneous 3×3 matrix `oMat2D` and a vector `oErr` of size $k_{max} \times 1$ that contains the error value Σ^2 between corresponding points in each step of the ICP.

Verify the function by transforming **all points** of a shape using an arbitrary 2D affine transformation and apply ICP to the two point sets. The obtained transformation `oMat2D` should be similar to the 2D affine transformation used to transform the points, while the final error value in `oErr` should be close to 0. Load reference and moving points of two different shapes and verify that the function adequately aligns the two shapes.

Homework Assignments

Homework report in the form of a Python script entitled `NameSurname_Exercise6.py` should execute the requested computations and function calls and display requested figures and/or graphs. It is your responsibility to load library packages and provide supporting scripts such that the script is fully functional and that your results are reproducible. The code should execute in a block-wise manner (e.g. `#%% Assignment 1`), one block per each assignment, while the answers to questions should be written in the corresponding block in the form of a comment (e.g. `# Answer: ...`).

1. Naložite poljubno obliko iz dane zbirke in načrtajte ustrezno zaporedje 2D linearnih preslikav tako, da se bo oblika rotirala okoli svojega središča. Prikažite delovanje preslikave tako, da obliko rotirate od $0 - 360^\circ$ s korakom 5° ter hkrati prikazujete točke oblike v istem prikaznem oknu.

2. Izvedite poravnavo vseh 2D oblik [0,30] v datoteki `letala.npy` na obliko z indeksom 0 z metodo iterativno najbližje točke, pri tem pa uporabite **2D afino aproksimacijsko poravnavo**. Izrišite v eno sliko vse 2D oblike v začetni legi, v drugo sliko pa vse oblike v poravnani legi.

3. **Toga poravnava**, ki se pogosto uporablja v praksi, je vedno aproksimacijska, saj zahtevano minimalno število parov korespondenčnih točk vodi do predoločenega sistema enačb – dobimo namreč več enačb kot pa je neznanih parametrov. Za poravnavo 2D oblik potrebujemo $K \geq 2$ korespondenčnih točk, preslikavo pa določimo z minimizacijo povprečne kvadratno Evklidske razdalje med točkami:

$$\Sigma^2 = \frac{1}{K}\sum_{i=1}^{K}\left[(x_i\cos\alpha - y_i\sin\alpha + t_x - u_i)^2 + (x_i\sin\alpha + y_i\cos\alpha + t_y - v_i)^2\right] ,$$

   ki jo odvajamo po parametrih t_x, t_y in α in odvode postavimo na nič. Rešitev sistema enačb je:

$$\alpha = -\mathrm{atan}\,\frac{\overline{uy} - \overline{vx} - \overline{u}\,\overline{y} + \overline{v}\,\overline{x}}{\overline{ux} + \overline{vy} - \overline{u}\,\overline{x} - \overline{v}\,\overline{y}} ,$$

$$t_x = \overline{u} - \overline{x}\cos\alpha + \overline{y}\sin\alpha ,$$

$$t_y = \overline{v} - \overline{x}\sin\alpha - \overline{y}\cos\alpha .$$

   Napišite funkcijo, ki med pari korespondenčnih točk določi 2D togo aproksimacijsko poravnavo:

```
def mapRigid2D( iPtsRef, iPtsMov ):
        return oMat2D
```

   kjer sta `iPtsRef` in `iPtsMov` matriki s koordinatami referenčnih in premičnih točk. Obe matriki imata dimenzije $K \times 2$, pri čemer je $K \geq 2$. Funkcija naj vrne 2D togo preslikavo v obliki homogene matrike `oMat2D` z dimenzijami 3×3. Prikažite delovanje funkcije tako, da **vse točke** izbrane oblike preslikate s poljubnimi parametri 2D toge preslikave in preverite ali funkcija vrne preslikavo v `oMat2D`, ki je enaka vaši 2D togi preslikavi.

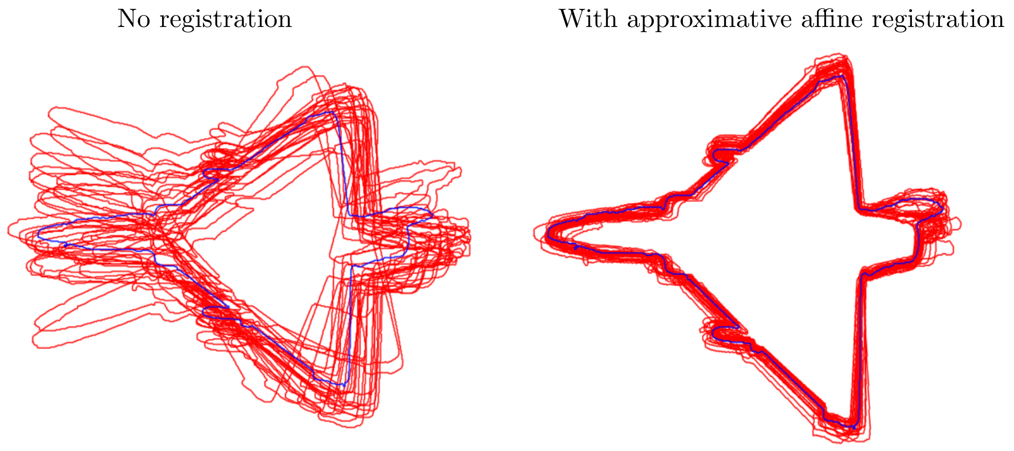
4. Izvedite poravnavo vseh 2D oblik [0,30] v datoteki `letala.npy` na obliko z indeksom 0 z metodo iterativno najbližje točke, pri tem pa uporabite **2D togo aproksimacijsko poravnavo**. Izrišite v eno sliko vse 2D oblike v začetni legi, v drugo sliko pa vse oblike v poravnani legi.

5. Napišite funkcijo, ki ustvari 4×4 matriko poljubne 3D afine preslikave:

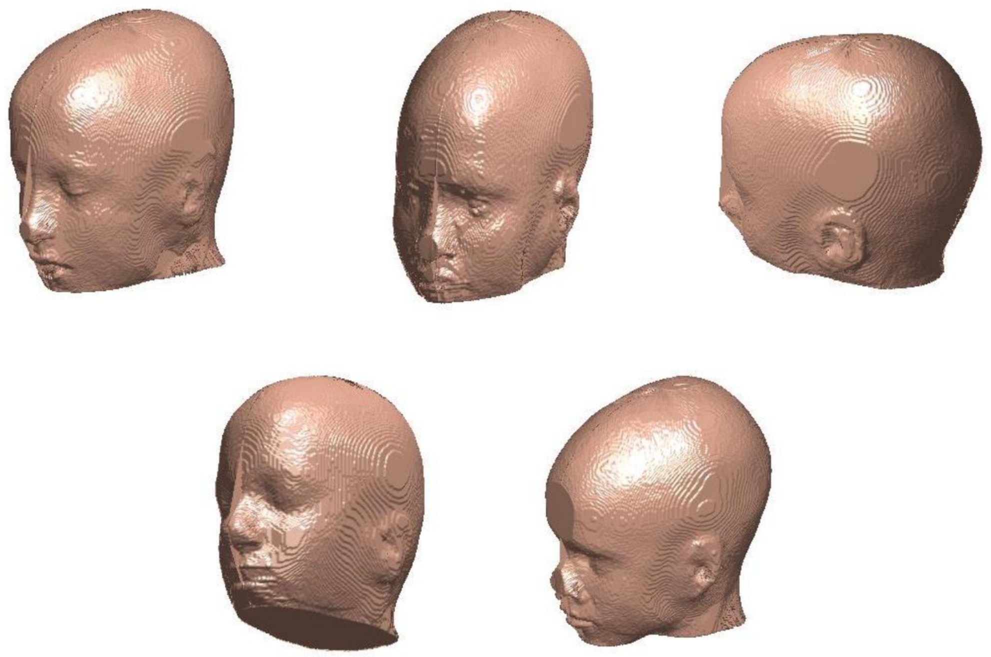
```
def transAffine3D( iScale, iTrans, iRot, iShear ):
        return oMat3D
```

kjer je `iScale` trivrstični vektor s parametri skale k_x, k_y in k_z, `iTrans` trivrstični vektor s parametri translacije t_x, t_y in t_z, `iRot` trivrstični vektor s koti rotacije α, β in γ v stopinjah in `iShear` šestvrstični vektor s parametri striga g_{xy}, g_{xz}, g_{xy}, g_{zx}, g_{zy}, g_{yz}. Funkcija vrne homogeno matriko `oMat2D`, ki ima dimenzije 4×4. Preverite delovanje funkcije na 3D oblikah v datoteki `glave.npy`[1] tako, da točke izbrane oblike preslikate s poljubnimi parametri striga, rotacije, translacije in skaliranja, nato pa še s poljubno 3D afino preslikavo. Vhodne in preslikane točke oblike prikažite z dano funkcijo `renderSurface.py`.

Transformations of 2D shape onto reference shape (*blue line*)



Arbitrary affine transformations of 3D shape



[1]Vseh pet 3D oblik naložite z ukazom *glave = numpy.load('glave.npy')* in so zapisane v seznamu tipa *list*, točke prve oblike pa dobite z ukazom *x=glave[0]*.