Exercises for Probabilistic Graphical Models
Sheet No. 5

Paul Swoboda, Gerard Pons-Moll, Jan-Hendrik Lange, Mohamed Omran

Due Date: 4th January 2019

Hand in: before 11:59pm by email to (mohomran at mpi-inf mpg de). Begin the subject of your e-mails with [PGM]. You should specify your first and last name as well matriculation number on submission. Please send your source code and all your results in a short report as a pdf.

In this exercises you will learn how to linearly align two sets of 2D or 3D point clouds in closed form using procrustes analysis. The exercises are prepared in the folder *06_BodyModels1/* and *06_BodyModels2/* .There, you will find the scripts that you need to run to answer the questions below. The functions you need to run the scripts are in the folder *06_BodyModels1/solution/* and *06_BodyModels2/solution/*. Those functions contain only the headers and you will have to complete them. Always use empirical or theoretical evidence to support your answers to the questions below. Include plots, graphs and figures where necessary.

Practical note: In order to better visualize the triangulated meshes, please install mayavi (*http://docs.enthought.com/mayavi/mayavi/*). We recommend also the usage of the ipdb debugger (*https://pypi.python.org/pypi/ipdb*). You can install it in Linux from the terminal running:

```
pip install mayavi ipdb
```

Note that *mayavi* requires the library *vtk* which you can install via `apt-get` or download it here: *https://www.vtk.org/download/*

For part 3 and 4 you will need the library *chumpy*:

```
pip install chumpy
```

1 Aligning points in 2D with Procrustes

Tasks: (1 point) Using the functionality in `06_BodyModels1/solution/procrustes.py`, complete the function in `06_BodyModels1/align_2D_points.py` and ...

- Find the optimal alignment between two ellipses. Debug until you achieve a good result!
- Try different ellipse rotations and scales. Does the method find the optimal solution for any scale, rotation and translation?

2 Aligning points in 3D with Procrustes

Find the rotation and translation that optimally aligns two sets of 3D points.
`06_BodyModels1/align_3D_points.py`

Tasks: (6 points)

- Implement a function that applies a random transformation to a 3D point cloud. -> `06_BodyModels1/solution/ra`
- Try the procrustes algorithm on 3D point clouds by running `align_3D_points.py`. Debug until you achieve good results!

`06_BodyModels1/align_3D_points_Gaussian_noise.py`

- Try different levels of Gaussian noise.
- Fix the noise to 0.005. Try different rotation and scales. Does it find the optimal solution for any scale, rotation and translation?
- For a given transformation, plot the error vs different levels of Gaussian noise, $sigma = [0 - 0.01]$. The error is defined as the sum of squared distances between points transformed source and target points (E in the course slides). How does the error increase with noise? Is the method robust to noise? Quantify the error with respect to the point cloud **before** you added the noise!
- Modify the Gaussian noise distribution: With probability P add the vector $sigma \cdot (1, 1, 1)^T$ as outlier to the vertex i. Set $sigma$ to 0.5 and visualize the resulting alignment.
  1. Plot the error vs the probability P, P ranging from 0 to 1.
  2. How do the alignments degrade with outliers?
  3. Is the method robust against this sort of noise? Why?
  4. How would you modify the method to make it robust to such outliers?
  5. With probability 0.2 plot the error against different $sigma$ levels.

`06_BodyModels1/align_3D_points_Gaussian_two_different_shapes.py`

- Does procrustes produce a satisfactory alignment between the two shapes? Why?

`06_BodyModels1/align_3D_points_Gaussian_two_different_poses.py`

- Does procrustes produce a satisfactory alignment between the two shapes with different poses? Why?
- What works better, aligning two different shapes in the same pose or two different poses of the same shape? Why?

`06_BodyModels1/align_3D_points_Gaussian_two_different_pose_shape.py`

- Try the procrustes code on two different shapes and different poses.
- How would you modify the method to align articulated structures like the human body?

`06_BodyModels1/align_3D_points_per_part.py`

- Implement a function that computes a per part procrustes alignment. Try the implemented function on the examples shown above, two different poses, two different shapes etc... What shortcomings does a part based alignment have? Reason in terms of robustness to noise, efficiency, accuracy, satisfaction of kinematic constraints.

3 Aligning points in 3D with the Iterative Closest Point Algorithm (ICP)

From here on you are going to work in the directory `06_BodyModels2/`. The code stubs may contain the keyword *TODO* which you will have to replace with working code.

Tasks: (6 points)

- Implement the two main parts of the ICP algorithm in `06_BodyModels2/solution/icp.py` corresponding to the condition `distance_Y_to_X=False`, around line 66. This condition corresponds to the algorithm as described in the lectures, meaning that the correspondences are computed from all points on the static mesh to the best points in the transformed mesh.

$$y_i^{j+1} = \arg\min_{y \in Y} \left\| f^j(y) - x_i \right\|^2$$

  Note that x and y are swapped in the lecture and the example code. Run `06_BodyModels2/rigid_align_without_correspondences.py` with `distance_Y_to_X` set to `False`, different random seeds and different amount of noise for the random transform. Does the algorithm converge always to a good solution? Why?

- Implement the two main parts of the ICP algorithm in `06_BodyModels2/solution/icp.py` corresponding to the condition `distance_Y_to_X=True`, around line 55. Under this condition, the correspondences should be computed from all points on the transformed mesh to the best points in the static mesh.

$$x_i^{j+1} = \arg\min_{x \in X} \left\| f^j(y_i) - x \right\|^2$$

  The code changes with respect to the previous task should be minimal. Before running the code, think about what do you expect to change.
  Run `06_BodyModels2/rigid_align_without_correspondences.py` with `distance_Y_to_X` set to `True`, different random seeds and different amount of randomness (parameter stds) for the random transform. Does the algorithm behave similarly or differently with this new distance computation? In which way are the solutions biased? Can you find a theoretical explanation for it?

4 Aligning points in 3D with ICP and Gradient Descent

(4 points)

- The function in `06_BodyModels2/solution/known_corr_gradient.py` solves the alignment problem with known correspondences with a gradient descent approach instead of procrustes. Read the code and complete the declaration of the p2p objective. Run `06_BodyModels2/rigid_align_with_correspondences.py` and play with different random seeds and transformation randomness (stds). How does it converge? How does it compare with your procrustes experiences in the first tasks?

- Let's try without correspondences. Read the function in `06_BodyModels2/solution/point2point.py`, which implements the same function as the one defined by you in the last task but computing correspondences whenever the points move. Fill the two arguments of the `point2point_squared` function in the correct order following the same `distance_Y_to_X` convention as in part 3. Run `06_BodyModels2/rigid_align_without_correspondences_gradient.py` and study the differences when changing the parameter `distance_Y_to_X` from True to False. Play with the random seed and amount of randomness in the transformation.?

5 Bonus

(1 point)

- Add a prior to the objective that penalizes the scale variations from its initialization value (the quotient of standard deviations).