Exercise 4

Linear Transformations and Projections

IAML 2020

Purpose

These exercises aim to introduce linear transformations and their implementation in Python. You will develop Python scripts to

- Apply geometric transformations in two-dimensions.
- Combine transformations (rotation, translation, and scale).
- Map points between two distinct planes using homographies.

Notes

- In the exercises based on script (i.e. `.py`) files, you are expected to implement code at certain points in the file. Although it is typically clear from context which piece of code you have to change, we have included specifier comments of the type `# <Exercise x.x (n)>`, where `x.x` is the exercise number and `(n)` is the letter associated with the specific task.
- Exercises marked by ^{extra} should only be attempted after completing the other exercises. The mark is added to indicate that the exercise is not essential but still important.

Exercise 4.1

*(**Jupyter**) Geometric transformations*

This exercise is written as a Jupyter notebook. It is saved as `Transformations.ipynb` *in the exercise material.*

In this exercise, you will create matrices and apply linear geometric transformations in vectors and points on the homogeneous and Euclidean coordinate systems. We cover the following topics:

1. Rotations and translations using homogeneous coordinates.
2. Bases and transforming points between them.

Exercise 4.2

Plane mapping using homographies

In this exercise you will use a homography to map coordinates between two planes under perspective projection. The goal is to map pixel coordinates of a person walking in the atrium at ITU to an overview map. We provide the the tracking coordinates for the person in the video but you have to create the homography and perform the map. Use the file `homography.py` to answer this exercise.

In the following, the the video `ITUStudent.mov` is denoted V, the ground floor in V is denoted G (ground floor) and the map of ITU building is denoted M as shown in Figure 1. The transformation from the ground floor G to the overview map M will be defined by a homography H_G^M.

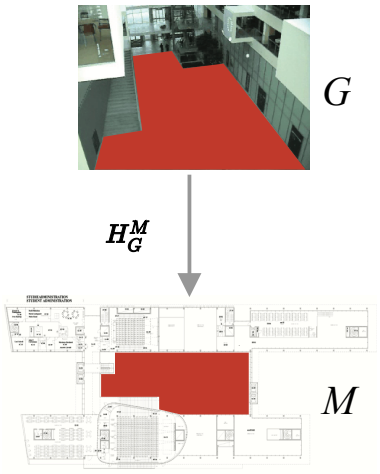

Figure 1: Naming the planes and transformations.

Overview

The script has a `main()` function which loads the relevant data and then plays the video while showing the overview map. You will have to add functionality to estimate a homography and use it to display the person's position on the overview map.

The map, video, and tracking data are loaded in the function `load_data()`. The tracking data is loaded from `trackingdata.dat` where each row r_i in the file is associate to a frame f_i in the video `ITUStudent.mov` and contains the coordinates of 3 rectangles (see Figure 2). The rectangles have coordinates pairs (x_1, y_1) and (x_2, y_2) which are the top left and bottom right corners of the rectangle. We save the rectangle points in arrays in a dictionary `data` with entries for *body*, *legs*, and *all*.

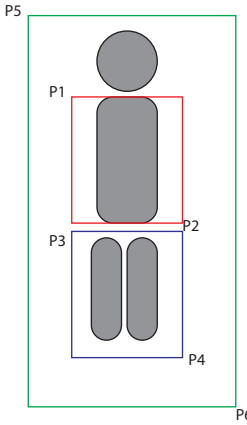

Figure 2: Example of three sub-regions of the person being tracked.

Task 1: *Creating the homography*

At least four corresponding points are needed to estimate the homography. In these exericises you will select the points P_i^G and P_i^M from the video and map manually via mouse clicks.

The function `load_or_create_homography()` automatically saves and loads a homography if possible, relieving you of having to set the points every time you run the script. You will implement the homography estimation in the else branch inside this function and add code to save and load the homography.

1. Use the helper function `get_points_from_mouse()` to get corresponding points in the video and the overview map. We provide a frame from the video `image_ground` and the map `image_map` for you to use. Read the docstring and comments of `get_points_from_mouse()` to understand how it works.

2. Use the function `cv2.findHomography(srcPoints, dstPoints)` [1] to estimate a homography from the points returned from `get_points_from_mouse()`.

[1] In depth tutorial about use of the [illegible] and about homographies in general.

3. Save the homography H_G^M in a file called `homography.npy`. Use the function `np.save()` [2] to save the Numpy array in a file.
4. Add code to load the homography if the file already exists using `load()` [3]

[2] A Numpy function to save an array to a binary file in NumPy .npy format.

[3] A Numpy function to load .npy files.

Task 2: *Estimate points on the overview map*

In this task you will use your homography to display the path walked by a student in the overview map. You have to map points that are on the ground floor G to the overview map M (i.e. use the points in `data['legs']`). all the entries in `data` contain points (x_i, y_i, x_i, y_i) in a $N \times 4$ array, i.e. the points for the ith frame can be accessed using `data[i]`.

1. Implement the function `get_center(part, i)`. It should return the center point of `part` at frame `i` in homogeneous coordinates (we included a helper function `to_homogeneous()`).
2. Implement the function `apply_homography(h, point)`. It should transform `point` in homogeneous using homography `h`. It should return the result as a euclidean coordinate (use `to_euclidean()`.
3. In the `main()` function, use your newly implemented functions to draw the center of the legs in the map view. Simply draw to `image_map` using one of OpenCV's drawing functions.
4. Do the same as in the previous questions but where you use an estimate of the head position.
5. Explain why it is important to use points on the feet and not the head when using a homography.
6. **Save the results:**$^{(Extra)}$ Save a video with the red, green and blue rectangles in the ground floor video. Save a video with the path walked by the person in the overview map. Save a figure (PNG) with the entire path walked by the user in the overview map.

Exercise 4.3

Reasoning about perspective and affine transformations

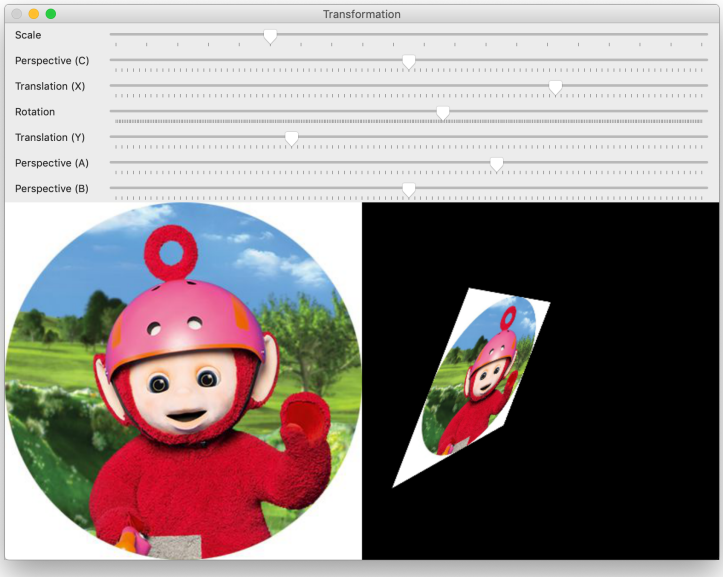

Figure 3: Example of multiple geometric transformations applied in an image.

This exercise is about improving your understanding of the effect of transformations. We provide a complete application (in `image_transformations.py`) for you to experiment with homographic transformations on images. You just have to answer some important questions on the effect of changing various parameters.

We use a general homography for this exercise. You will be able to just translation, scale, rotation, and perspective parameters. To enable this, we decompose the transformation matrix in the following manner:

$$\begin{bmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ p_1 & p_2 & p_3 \end{bmatrix}, \tag{1}$$

where s is the scaling factor, θ is the rotation in radians, t_x, t_y is the translation vector, and p_1, p_2, p_3 controls perspective factors (more about this later).

Task 1: *Experiment*

Use the application to get a feel for how the different parameters affect the transformation.

Task 2: *Reason about perspective and affine transformations*

This exercise will make you think closely about the effect of parameters on the transformed image and how the perspective transform

works. Use the following questions to determine how well you understand the concepts.

1. Usually, we use homogeneous coordinates $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ for a 2d point $\begin{bmatrix} x \\ y \end{bmatrix}$. Imagine what would happen if you used a different number than 1, let's say an arbitrary number k. Explain the effect of k on the transformed coordinates x', y'. Try to write out the complete calculation of

$$\begin{bmatrix} x' \\ y' \\ \lambda \end{bmatrix} = \begin{bmatrix} m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 \\ m_7 & m_8 & m_9 \end{bmatrix} \begin{bmatrix} x \\ y \\ k \end{bmatrix} = ? \tag{2}$$

2. Try to explain why k is always set to 1 for homogeneous transformations?

3. In the matrix at the beginning of the exercise, what effect does changing p_3 have on the transformed image? Why is this the case?

4. Similarly, explain what effect p_1 and p_2 have. It might help to calculate an example using pen and paper. Don't forget to perform the conversion to euclidean coordinates.

5. What can you say in general about the relation of the z coordinate in homogeneous coordinates (usually shown as the scaling factor λ) to the creation of the perspective effect?

Exercise 4.4

3D Transformations^{extra}

In this exercise, you will work with linear geometric transformations of the in a cube in three-dimensional Euclidean space. Use the file `cube_rotation.py` to answer the following questions.

A tree-dimensional cube is an object formed by eight vertices, as show in Figure 4.

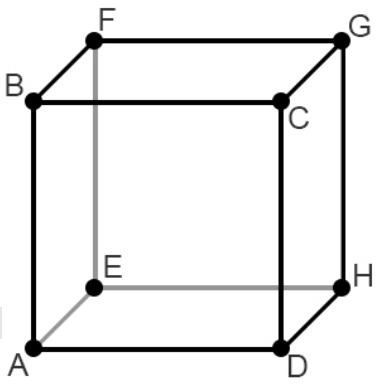

Figure 4: The vertices of a three-dimensional cube.

1. **Create a 3D cube:** At the bottom of the file, you will find the setup code for the cube and window. Identify the where `verticesCube` defines the coordinates of the 3D cube. The subsequent line in the calls (`draw_3d_cube()`) to draw the 3D cube. The Matplotlib window has 3 sliders called `x_slider_theta` , `y_slider_theta` , and `z_slider_theta` that are used to change the rotation angles (in degrees) on X-, Y-, and Z-axes.

2. **Create the rotation matrices:** Make a function `get_3d_rotation_matrix(theta, axis)` , which returns the 3D rotation matrix `R_i` given an input angle `theta` (in radians) and an axis ID (0: X-axis, 1: Y-axis, 2: Z-axis),

3. **Rotating multiple vertices:** Use the the function `update_3d_cube()` to rotate to rotate the vertices of the cube when the user changes a *slider*. Notice:

   - The angle θ is in degrees, but Python uses angles in radians to apply geometric transformation. Use the function `np.radians()` to convert the angle θ;
   - The rotations on X-, Y-, and Z-axes are cumulative. Remember to apply the first geometric transformation in the vertices you have defined;
   - Show the rotated cube using the function `draw_3d_cube()` , then test your code.