Exam Part 1 report
Course: Introduction to Image Analysis and Machine Learning, BSc + Machine Learning, MSc (Spring 2020)
BSIIAML1KU

Jonas Ishøj Nielsen

13/03/2020

Contents

1 Task 2: Report

1.1 Approach

The transformation.py file contains a class I named Transformation, that contains the transformation in a variable called M, because it followed the books naming convention. I did it this way so that transformations can be chained and because it would reduce methods shown to anyone using the api opposed to if there weren't any classes.
The generations are outside the class as python doesn't really support multiple constructors. To generate a transformation I have added the functions "identity_3x3()" that creates a 3x3 identity matrix. I have also added the function "transformation_3x3" that create a custom transformation. It is heavily inspired by the function "perspective_transform" from exercise 4. Through parameters rotation, translation and scaling can be set. I kept the parameter for setting perspective, as it might be usefull later, but it is last have have default values 0,0,1, so it doesn't impact unless specified. The function "transformation_arbitrary_3x3" create a transformation where the object will first be translated then rotated then translated back.
The class Transformation contains method like "inverseOfM" that use numpy's inverse function to return the inverse of the transformation. The function "transpose" returns the transposed transformation, "pseudoinverse" returns the pseudo inverse and is done by using numpy's pinv function. The function have commented out code that would return the pseudoinverse calculated by using the formula: "(self.T * self).inverse * self.T" but I commented it out as it fails if "self.T * self" is a singular matrix.
The "test" function in the file "test_transformations.py" contains multiple examples of creating, combining and applying transformations.

1.2 Implementation details

The funciton "combineMultiple" takes a list of transformations and combine them in order given, using the Transformation class's combine method. The combine method uses numpy's dot method and is made so one can for transformations t1,t2,...,tn write t1.combine(t2).combine(t3).....combine(tn).
There are 3 functions that apply transformation to points as I thought it was unclear what type of parameter it should take. The funcitons are: "transformationPointApply_3x3", "transformationMultiplePointsApply_3x3" and "transformationMultipleColumnPointsApply_3x3". The difference is what type of points it takes which is explained in the Multi-line Docstrings. No matter what function all multiplies the transformation with the points and returns an list with to columns, first being for x values and second for y values.
There are 2 functions that apply transformation to images. "transformationImageApply_3x3" uses cv2.warpPerspective. The second is "transformationImageApply_3x3_My" that uses backwards mapping, and takes details for a rectangle. This function was added to handle Task 2's Effective transformations

improvement, but isn't the primary one as it is really slow for reasons I am unsure of how to fix.

1.3 Image tests

The function "image_test" in "test_transformations.py" contains the code for the 2 transformations t1 and t2, the code for applying to the image and the code to save to a file, and a test to reload and show the transformed images.

Figure 1: Original image

Figure 2: Transformation t1 applied

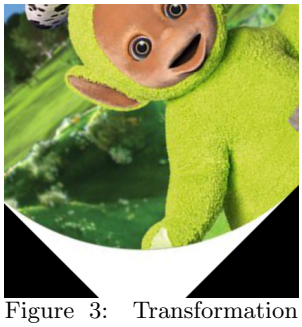

Figure 3: Transformation t1 applied

1.4 Manual tests

Below in figure 4 is the manual calculation for transformation t1. This trans-

$$m_3m_2m_1 = \begin{bmatrix}1 & 0 & -300\\ 0 & 1 & -250\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}3 & 0 & 0\\ 0 & 2 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos(\pi/8) & -\sin(\pi/8) & 0\\ \sin(\pi/8) & \cos(\pi/8) & 0\\ 0 & 0 & 1\end{bmatrix}$$

$$m_3m_2 = \begin{bmatrix}1*3+0+-300*0 & 1*0+0+-300*0 & 1*0+0+-300*0\\ 0+1*0+-250*0 & 0+1*2+-250*0 & 0+1*0+-250*0\\ 0+0+1*0 & 0+0+1*0 & 0+0+1*1\end{bmatrix} = \begin{bmatrix}3 & 0 & -300\\ 0 & 2 & -250\\ 0 & 0 & 1\end{bmatrix}$$

$$m_3m_2m_1 = \begin{bmatrix}3 & 0 & -300\\ 0 & 2 & -250\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos(\pi/8) & -\sin(\pi/8) & 0\\ \sin(\pi/8) & \cos(\pi/8) & 0\\ 0 & 0 & 1\end{bmatrix}$$

$$= \begin{bmatrix}3*\cos(\pi/8)+0*\sin(\pi/8)+-300*0 & 3*-\sin(\pi/8)+0*\cos(\pi/8)+-300*0 & 3*0+0*0+-300*1\\ 0*\cos(\pi/8)+2*\sin(\pi/8)+-250*0 & 0*-\sin(\pi/8)+2*\cos(\pi/8)+-250*0 & 0*0+2*0+-250*1\\ 0*\cos(\pi/8)+0*0+1*0 & 0*-\sin(\pi/8)+0*0+1*0 & 0*0+0*0+1*1\end{bmatrix}$$

$$= \begin{bmatrix}3*\cos(\pi/8) & 3*-\sin(\pi/8) & -300\\ 2*\sin(\pi/8) & 2*\cos(\pi/8) & -250\\ 0 & 0 & 1\end{bmatrix}$$

Figure 4: Calulation of transformation t1

formaito can then be added to a homogeneous point [[x],[y],[1]] like in figure 5 which when x = 12 and y = 55, gives same result as the function "point_test".

$$m_3 m_2 m_1 \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 3 * cos(\pi/8)*x + 3 * -sin(\pi/8)*y - 300 \\ 2*\sin(\pi/8)*x + 2 * \cos(\pi/8)*y - 250 \\ 1 \end{bmatrix} = \begin{bmatrix} 2.7716386x + -1.1480503y - 300 \\ 0.765366865x + 1.84775907y - 250 \\ 1 \end{bmatrix} \text{-- > } \begin{bmatrix} 2.7716386x + -1.1480503y - 300 \\ 0.765366865x + 1.84775907y - 250 \end{bmatrix}$$

$$m_3 m_2 m_1 \begin{bmatrix} 12 \\ 55 \\ 1 \end{bmatrix} = \begin{bmatrix} 2.7716386*12 + -1.1480503*55 - 300 \\ 0.765366865*12 + 1.84775907*55 - 250 \end{bmatrix} = \begin{bmatrix} -329.8831033 \\ -139.18884877 \end{bmatrix}$$

Figure 5: Apply t1 to a homogeneous point

1.5 Checklist

- ☑ The file transformations.py with your library implementation.
- ☑ The file test_transformations.py with your test code.
- ☑ Report section as specified
- ☑ Two pairs of before/after images

2 Task 2: Report

2.1 Tests

The function "reportTest_2_1" transform 3 points, then infer the affine transformation using the function. I have added automatic tests to show that it gives the same output as original, and contains the same values for the unknowns.

2.2 Checklist

- ☑ The file transformations.py with the learn_affine() function added.
- ☑ The file test_affine.py with your test code.
- ☑ The specified report sections.

3 Task 3: Report

3.1 Evaluation of initial implementation

My implementation works smoothly and I haven't encountered anything I would label as unintended behavior or issues. Below is the three examples. And the recording is in vid_1.mp4.

3.2 On improvements

I have implemented Caching and Effective transformations, although the Effective transformation is only used if the user manually changes the boolean value on line 212 in the file "texture-mapping.py" from False to True.

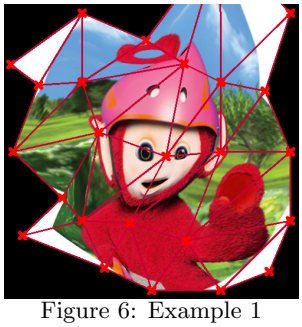
Figure 6: Example 1

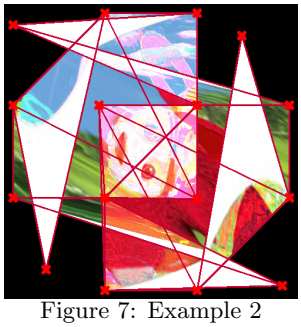
Figure 7: Example 2

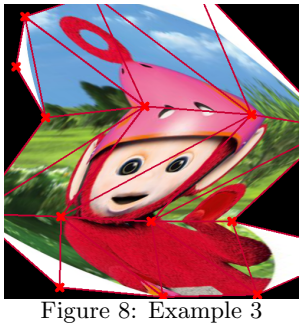
Figure 8: Example 3

3.2.1 Caching

I use the class TextureMap's value Cache to store a list of known patches. In update if the indices is None then cached becomes a new list with each value being None, else the effected indices is set to None. In the function "_update_patches" if an indice isn't None then the patched is set to that patch. Else that part is transformed and stored in the patches and the cached.
I have added a print statement to say which is updated and it shows it works. It definitly makes it faster and it is necessary to use if one chooses to run using the Effective transformations. I added an implementation where the indices in printed in the bottom of the screen for a small duration and it can be seen in vid_2.mp4. The video clearly show which indices is updated, and which aren't.

3.2.2 Effective transformations

I have done this by making a function "transformationImageApply_3x3_My" in the file "transformations.py". It is quire slow but only transform pixels within the range and uses backward-mapping. This wasn't the original approach, but all my other attempts to cut of a part failed. Video vid_3.mp4 shows it in use.

3.3 Checklist

3.4 Checklist

- ☑ The file texture-mapping.py with all the changes you made, i.e. you do not need to create a separate file for the improvements task.
- ☑ Three example pictures of the application in action.
- ☑ A video demonstrating that the application works.
- ☑ A video demonstrating the improved version of the application.
- ☑ Report section as specified.