

1.3 Technical Task (80 points)

1. (Given) In lab, we worked with a program to control rotation of the image using a slider.

2. Implement Scaling. Part 0 of the lab is to implement the function getScaleMatrix to create

a matrix representing a scaling operation. The provided skeleton code has event handling

that will call your function in order to produce a result. Drag the slider to produce a scaled

version of your image and save your result as "Assignment5 Part0 Output Scale.png" (Do

not submit the image for rotation)



**Assignment5\_Part0\_Output\_Scale.png**

Scaled 320x240 Displayed @ 25%

**hand.png**

Original 640x480 Displayed @ 25%

3. Reproduce the OpenCV rotation function. Part 1 of the lab is to reproduce the OpenCV

getRotationMatrix2D function. To do this, you will \_rst implement the getTranslationMatrix

function to create a matrix representing a basic translation. Then, using the scaling operation

from Part 0, you will need to implement the function myGetRotationMatrix2D by correctly

chaining together the matrix primitives. The skeleton code will display the result of your

version along side the result of the OpenCV version. Save a rotated and scaled version of

your image as "Assignment5 Part1 Output Mine.png". Submit it along with the matching

"Assignment5 Part1 Output OpenCV.png".

Mat myGetRotationMatrix2D(Point2f center, double rotationAngle, double scaleFactor)

{

Mat mat1 = Mat::eye(Size(3,3), CV\_64FC1);

Mat mat2 = Mat::eye(Size(3,3), CV\_64FC1);

Mat mat3 = Mat::eye(Size(3,3), CV\_64FC1);

Mat mat4 = Mat::eye(Size(3,3), CV\_64FC1);

// Move image to origin

mat1.at<double>(0, 2) = -center.x;

mat1.at<double>(1, 2) = -center.y;

// Rotate image

double theta = rotationAngle \* M\_PI / 180.0;

mat2.at<double>(0, 0) = cos(theta);

mat2.at<double>(0, 1) = sin(theta);

mat2.at<double>(1, 0) = -sin(theta);

mat2.at<double>(1, 1) = cos(theta);

// Scale image

mat3.at<double>(0, 0) = scaleFactor;

mat3.at<double>(1, 1) = scaleFactor;

// Move back to original location

mat4.at<double>(0, 2) = center.x;

mat4.at<double>(1, 2) = center.y;

// Return composition of matrices

return mat4 \* (mat3 \* (mat2 \* mat1));

}





**Assignment5\_Part1\_Output\_Mine.png Assignment5\_Part1\_Output\_OpenCV.png**

4. Implement Shearing. Part 2 of the lab is to create functionality that is missing in OpenCV:

create a transformation where the image is sheared about its center. You will start by imple-

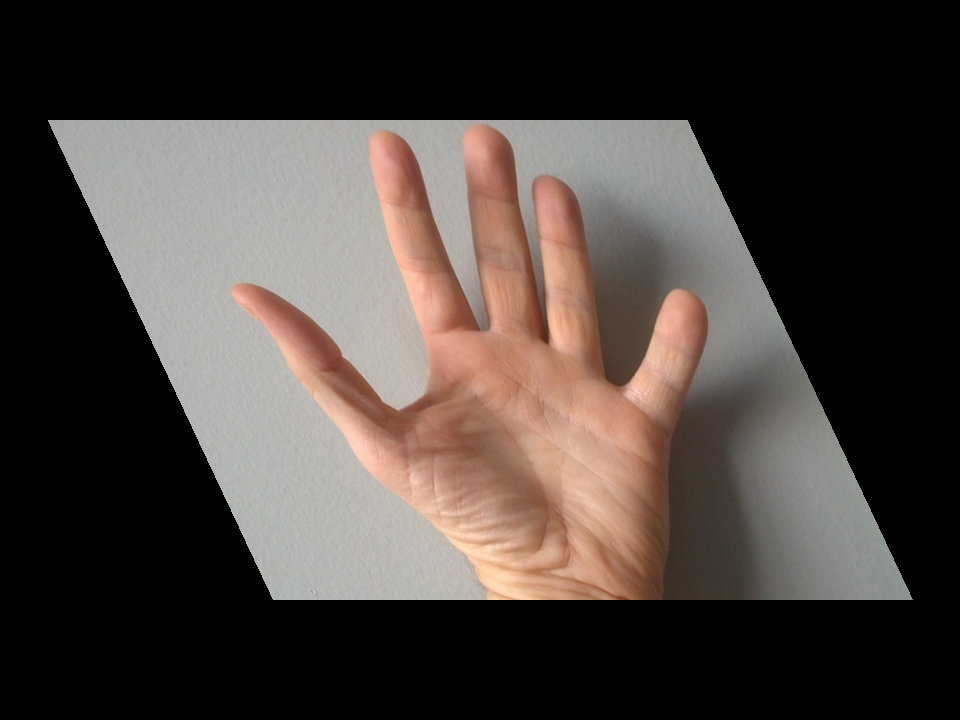
menting the primitive getShearMatrix. Then, in a style similar to Part 1, you will implement

the function getShearMatrix2D that composes several primitive transformations together to

shear the image about its center. Use the sliders to shear the image horizontally and vertically.

Save a horizontally sheared version as "Assignment5 Part2 Output Horizontal.png". Save a

vertically sheared version as "Assignment5 Part2 Output Vertical.png"



**Assignment5\_Part2\_Output\_Horizontal.png Assignment5\_Part2\_Output\_Vertical.png**

5. Create a spring-time collage by assembling a collection of at least 5 source images no more

than 350 x 350 pixels. Use any transformations you like to place the images with a 750 x 750

pixel canvas. Save your result as "Assignment5 Output.png".

The source images should be placed at di\_erent scales and orientations. You can use shears

too if you like. The source images should be scattered throughout the canvas (not clustered in the upper left). Each image may be used more than once if you would like. (If you don't

want to make a spring-themed collage, you can use subject matter of your choosing)

The skeleton code for this part is minimal in order to give you freedom to develop your own

solution from scratch. You can assemble your code using any pieces from any of the labs or

homeworks that we have done. You can make an interface to support your composition if

you want, but you don't have to. You can generate the con\_guration of the sub-images with

mathematical formula, by hand, or at random. No matter how you choose to construct your

placement, your source images should remain mostly within the canvas and be spread through-

out the canvas. An example composition is provided in "Assignment5 Output Diane.png".



**Assignment5\_Output.png**