January 2017
School of Engineering and Technology
Name: Teera Laiteerapong, 118950

Backward mapping

```
#include <iostream>
#include "opencv2/opencv.hpp"

#define PI 3.141592653589793

void pixelInterpolation(cv::Mat matImage, cv::Mat matOutput) {
  // Assume that the original image
  // has a shape similar to square
  // so that, we will get its width and height as below
  float output2Width = (sqrt(pow(matImage.rows, 2) + pow(matImage.cols, 2)));
  // We create a bigger size of image from the width we get above
  cv::Mat matOutput2(output2Width, output2Width, CV_8UC1);
  // Set center of a new image
  cv::Mat matTransMidBack(3, 3, CV_32FC1, cv::Scalar::all(0));
  matTransMidBack.at<float>(0, 0) = 1;
  matTransMidBack.at<float>(0, 1) = 0;
  matTransMidBack.at<float>(0, 2) = (matOutput2.rows - matImage.rows)/2;
  matTransMidBack.at<float>(1, 0) = 0;
  matTransMidBack.at<float>(1, 1) = 1;
  matTransMidBack.at<float>(1, 2) = (matOutput2.cols - matImage.cols);
  matTransMidBack.at<float>(2, 2) = 1;

  // cos sin 0
  // -sin cos 0
  // 0 0 0
  // 30 degree: PI/6 (180/6 = 30)
  float rotationRadian = PI/6;
  // Rotate the image back (-30 degree)
  cv::Mat matRotBack(3, 3, CV_32FC1, cv::Scalar::all(0));
  matRotBack.at<float>(0, 0) = cos(rotationRadian);
  matRotBack.at<float>(0, 1) = sin(rotationRadian);
  matRotBack.at<float>(1, 0) = -sin(rotationRadian);
  matRotBack.at<float>(1, 1) = cos(rotationRadian);
  matRotBack.at<float>(2, 2) = 1;

  // Rotate the image from the center of the image
  cv::Mat matTrans(3, 3, CV_32FC1, cv::Scalar::all(0));
  matTrans.at<float>(0, 0) = 1;
  matTrans.at<float>(0, 1) = 0;
  matTrans.at<float>(0, 2) = - matImage.rows / 2;
  matTrans.at<float>(1, 0) = 0;
  matTrans.at<float>(1, 1) = 1;
  matTrans.at<float>(1, 2) = - matImage.cols / 2;
  matTrans.at<float>(2, 2) = 1;

  // Here we're looping through
  // each point of the image output
  for (int i=0; i < matOutput.rows; i++) {
    for (int j=0; j < matOutput.cols; j++) {
      cv::Mat inputCoordinate(3, 1, CV_32FC1);
      // Y-axis
      inputCoordinate.at<float>(0, 0) = i;
      // X-axis
```

```
      inputCoordinate.at<float>(1, 0) = j;
      inputCoordinate.at<float>(2, 0) = 1.0;
      // First, we translate the point to the origin
      // of the rotation. Then, we do rotation.
      // Finally, we translate it back and set the image to the center
      cv::Mat tmp = matTransMidBack * matRotBack * matTrans * inputCoordinate;

      // Rotate only within matOutput size
      if (((tmp.at<float>(0, 0) >= 0) &&
          (tmp.at<float>(0, 0) <= matImage.rows) &&
          tmp.at<float>(1, 0) >= 0) &&
         (tmp.at<float>(1, 0) <= matImage.cols)) {
          matOutput2.at<unsigned char>(i, j) = matImage.at<unsigned char>(
            // add pixel value from original image
            // back to the rotated pixel value
            tmp.at<float>(0, 0),
            tmp.at<float>(1, 0));
      }
    }
  }

  // Write an interpolated and rotated image to a file
  cv::imwrite("./rotated_image.jpg", matOutput2);
}

void rotateImage(cv::Mat matImage) {
  // Create a new Mat that has 2 times size from the original image
  cv::Mat matOutput(matImage.rows * 2, matImage.cols * 2, CV_8UC1);

  // Use floating point data type
  // Rotation matrix
  // cos -sin 0
  // sin cos  0
  // 0    0   0
  cv::Mat matRot(3, 3, CV_32FC1, cv::Scalar::all(0));
  // 30 degree: PI/6 (180/6 = 30)
  float rotationRadian = PI/6;
  // Rotate 30 degree counterclockwise
  // The origin of the rotation
  // is on the top right of the original image
  matRot.at<float>(0, 0) = cos(rotationRadian);
  matRot.at<float>(0, 1) = -sin(rotationRadian);
  matRot.at<float>(1, 0) = sin(rotationRadian);
  matRot.at<float>(1, 1) = cos(rotationRadian);
  matRot.at<float>(2, 2) = 1;


  // Therefore, we need to translate the center
  // of the original image to the origin of the rotation
  cv::Mat matTrans(3, 3, CV_32FC1, cv::Scalar::all(0));
  matTrans.at<float>(0, 0) = 1;
  matTrans.at<float>(0, 1) = 0;
  matTrans.at<float>(0, 2) = - matImage.rows / 2;
  matTrans.at<float>(1, 0) = 0;
  matTrans.at<float>(1, 1) = 1;
```

```
  matTrans.at<float>(1, 2) = - matImage.cols / 2;
  matTrans.at<float>(2, 2) = 1;

  // After that, we need to translate it back
  cv::Mat matTransBack(3, 3, CV_32FC1, cv::Scalar::all(0));
  matTransBack.at<float>(0, 0) = 1;
  matTransBack.at<float>(0, 1) = 0;
  matTransBack.at<float>(0, 2) = matImage.rows / 2;
  matTransBack.at<float>(1, 0) = 0;
  matTransBack.at<float>(1, 1) = 1;
  matTransBack.at<float>(1, 2) = matImage.cols / 2;
  matTransBack.at<float>(2, 2) = 1;

  // Here we're looping through
  // each point of the original image
  for (int i=0; i < matImage.rows; i++) {
    for (int j=0; j < matImage.cols; j++) {
      cv::Mat inputCoordinate(3, 1, CV_32FC1);
      // Y-axis
      inputCoordinate.at<float>(0, 0) = i;
      // X-axis
      inputCoordinate.at<float>(1, 0) = j;
      inputCoordinate.at<float>(2, 0) = 1.0;
      // First, we translate the point to the origin
      // of the rotation. Then, we do rotation.
      // Finally, we translate it back
      cv::Mat tmp = matTransBack * matRot * matTrans * inputCoordinate;

      // Rotate only within original image
      if (((tmp.at<float>(0, 0) >= 0) &&
           (tmp.at<float>(0, 0) <= matImage.rows) &&
           tmp.at<float>(1, 0) >= 0) &&
          (tmp.at<float>(1, 0) <= matImage.cols)) {
            matOutput.at<unsigned char>(
              // add pixel value to the rotated pixel value
              tmp.at<float>(0, 0),
              tmp.at<float>(1, 0)) = matImage.at<unsigned char>(i, j);
      }
    }
  }

  // We need to get the missing pixel
  // by rotate matOutput back and compare the pixel
  // with the original image, then replace black pixel
  // with pixel value
  pixelInterpolation(matImage, matOutput);
}

int main(int argc, char *argv[]) {
  // Read image in grayscale
  cv::Mat matImage = cv::imread("./cameraman2.tif", cv::IMREAD_GRAYSCALE);
  // Do 30 degree rotation on the image
  rotateImage(matImage);
  return 0;
}
```