ITMO University

Faculty of Control Systems and Robotics

Report for laboratory work №2

course "Digital Image Processing"

Images Geometric Transformations

Done by:

Ha The Long Vuong
Group: R42334c

Supervised by:

Shavetov Sergey Vasilievich

Saint Petersburg
2021

Contents

List of Figures

Listings

1 Purpose

Studying of maping main types and using geometric transformations for spatial image correction.

2 Assignment

2.1 Linear Transformation

2.1.1 Image Shift

The new image obtained from *shift* transformation is

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} T \Rightarrow T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ C & F & 1 \end{bmatrix} \tag{1}$$

OpenCV code

Listing 1: Image shift using OpenCV and C++ programming language

```
void DIP::lab2::shift_image(const cv::Mat& img, cv::Mat& new_img) {
  cv::Mat T = (cv::Mat_<double>(2, 3) << 1, 0, 100, 0, 1, 150);
  cv::warpAffine(img, new_img, T, cvSize(img.cols, img.rows));
}
```

Result

Figure 1: Image shift using OpenCV and C++ programming

2.1.2 Image Flip

The new image obtained from *flip* transformation is

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} T \Rightarrow T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

OpenCV code

Listing 2: Image flip using OpenCV and C++ programming language

```
void DIP::lab2::flip_image(const cv::Mat& img, cv::Mat& new_img) {
  cv::Mat T = (cv::Mat_<double>(2, 3) << 1, 0, 0, 0, -1, img.rows - 1);
  cv::warpAffine(img, new_img, T, cvSize(img.cols, img.rows));
}
```

Result

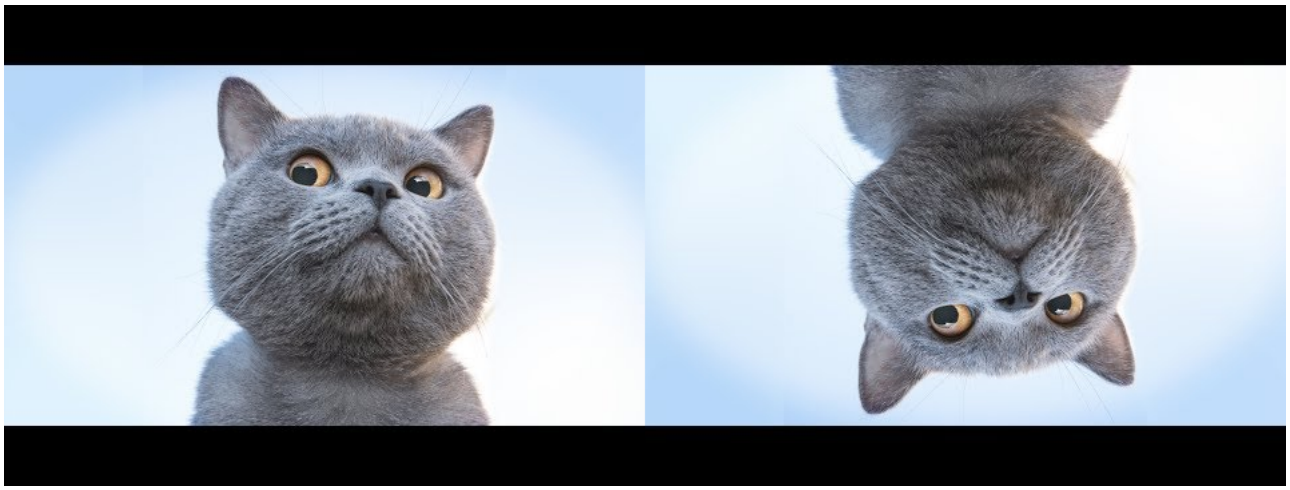

Figure 2: Image flip using OpenCV and C++ programming

2.1.3 Uniform Image Scaling

The new image obtained from *scale* transformation is

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} T \Rightarrow T = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

The size of new obtained image is depend on $\alpha\,\beta$

1. $\alpha < 1\,\beta < 1$, then new image will decrease size.
2. $\alpha > 1\,\beta > 1$, then new image will increase size.
3. $\alpha \neq \beta$, then new image will not be keep the ratio wight and height compare with original image.

OpenCV code

Listing 3: Image scale using OpenCV and C++ programming language

```
void lab2::uniform_scale(double scale, const cv::Mat& img, cv::Mat& new_img)
    {
  cv::Mat T = (cv::Mat_<double>(2, 3) << scale, 0, 0, 0, scale, 0);
  cv::warpAffine(img, new_img, T,
                 cvSize(int(img.cols * scale), int(img.rows * scale)));
}
```

Result

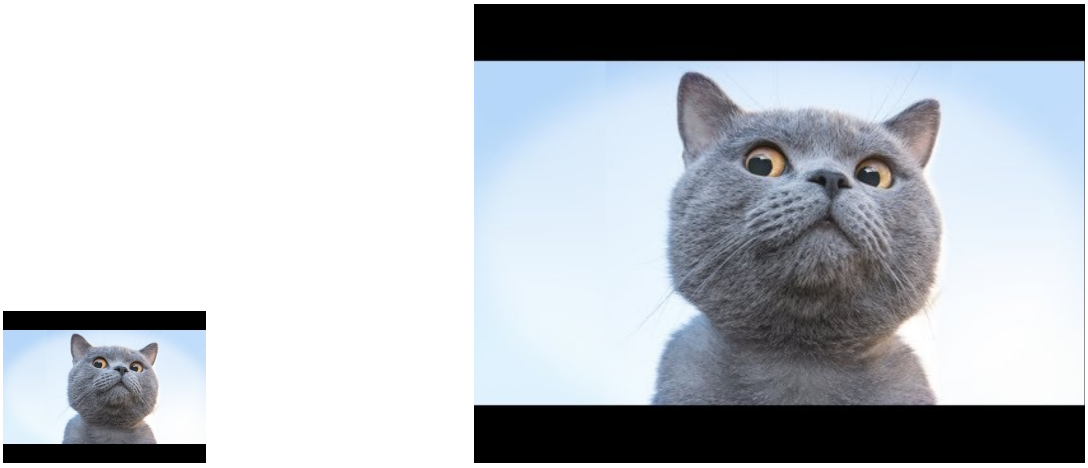

Figure 3: Image scale using OpenCV and C++ programming

2.1.4 Image Rotation

The new image obtained from *rotation* transformation is

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} T \Rightarrow T = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

where φ is the angle that image will rotate in *clockwise* direction.

OpenCV code

Listing 4: Image rotate using OpenCV and C++ programming language

```
void DIP::lab2::rotate_image(const double angle, const cv::Mat& img,
                             cv::Mat& new_img) {
  cv::Point center = cv::Point(img.cols / 2, img.rows / 2);
  cv::Mat T = cv::getRotationMatrix2D(center, angle, 1);
  cv::warpAffine(img, new_img, T, cvSize(img.cols, img.rows));
}
```

Result

Figure 4: Image rotation using OpenCV and C++ programming

2.1.5 Affine Mapping

Affine mapping — is a mapping, in which parallel lines go into parallel lines, intersect lines into intersect lines, cross lines into cross lines; the segments lengths ratios of lying one the one straight line or in parallel are preserved.

OpenCV code

Listing 5: Affine mapping using OpenCV and C++ programming language

```
void lab2::affine2d_image(const cv::Mat& img, cv::Mat& new_img) {
  cv::Point2f srcTri[3];
  srcTri[0] = cv::Point2f(0.f, 0.f);
  srcTri[1] = cv::Point2f(float(img.cols) - 1.f, 0.f);
  srcTri[2] = cv::Point2f(0.f, float(img.rows) - 1.f);
  cv::Point2f dstTri[3];
  dstTri[0] = cv::Point2f(0.f, float(img.rows) * 0.33f);
  dstTri[1] = cv::Point2f(float(img.cols) * 0.85f, float(img.rows) * 0.25f);
  dstTri[2] = cv::Point2f(float(img.cols) * 0.15f, float(img.rows) * 0.7f);
  cv::Mat warp_mat = cv::getAffineTransform(srcTri, dstTri);
  cv::warpAffine(img, new_img, warp_mat, new_img.size());
}
```

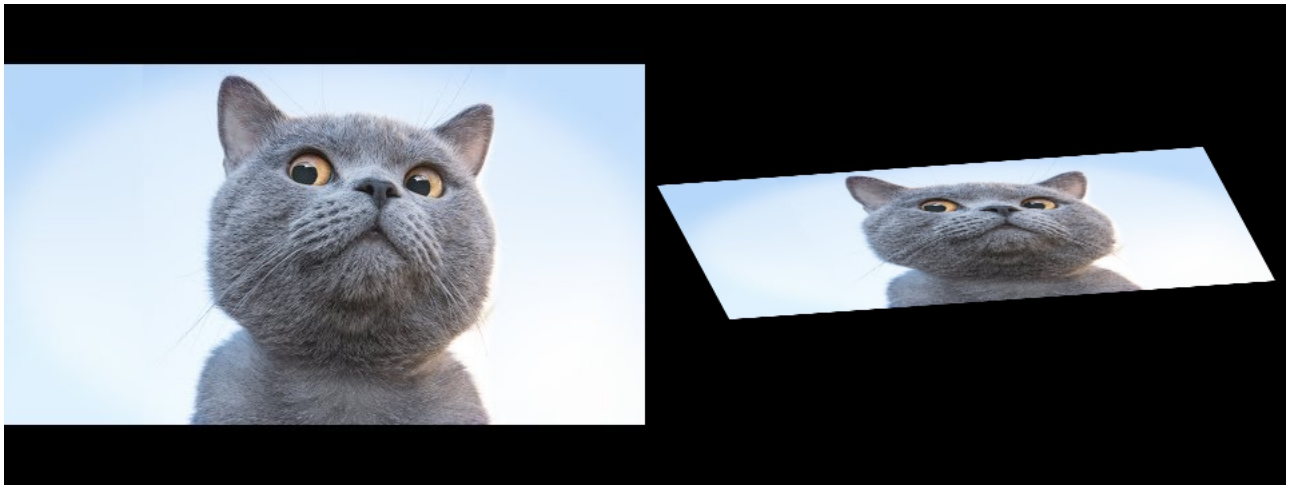

Figure 5: Affine mapping using OpenCV and C++ programming

2.1.6 Image Bevel

The new image obtained from *bevel* transformation is

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} T \Rightarrow T = \begin{bmatrix} 1 & 0 & 0 \\ s & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

OpenCV code

Listing 6: Image bevel using OpenCV and C++ programming language

```
void lab2::bevel_image(double cof, const cv::Mat& img, cv::Mat& new_img) {
  cv::Mat T = (cv::Mat_<double>(2, 3) << 1, cof, 0, 0, 1, 0);
  cv::warpAffine(img, new_img, T,
                 cvSize(int(img.cols + cof * img.rows), img.rows));
}
```

Result

Figure 6: Image bevel using OpenCV and C++ programming

2.1.7 Piece-wise-Linear Mapping

Piece-wise-Linear Mapping - is mapping in which the image is splitted into parts, and the various linear transformations are applied to each if these parts.

OpenCV code

Listing 7: Piece-wise-flip image using OpenCV and C++ programming language

```
void lab2::flip_pw(const cv::Mat& img, cv::Mat& new_img) {
  cv::Mat T = (cv::Mat_<double>(2, 3) << 1, 0, 0, 0, -1, img.rows - 1);
  cv::Mat temp_right;
  img(cv::Rect(int(new_img.cols / 2), 0, new_img.cols - int(new_img.cols /
   2),
               new_img.rows))
      .copyTo(temp_right);
  cv::warpAffine(temp_right, temp_right, T,
                 cvSize(int(temp_right.cols), temp_right.rows));
  img(cv::Rect(0, 0, int(img.cols / 2), img.rows))
      .copyTo(new_img(cv::Rect(0, 0, int(img.cols / 2), img.rows)));

  temp_right.copyTo(new_img(
      cv::Rect(int(new_img.cols / 2), 0, int(new_img.cols / 2), new_img.rows
   )));
}
```

Result

Figure 7: Piece-wise-flip image using OpenCV and C++ programming

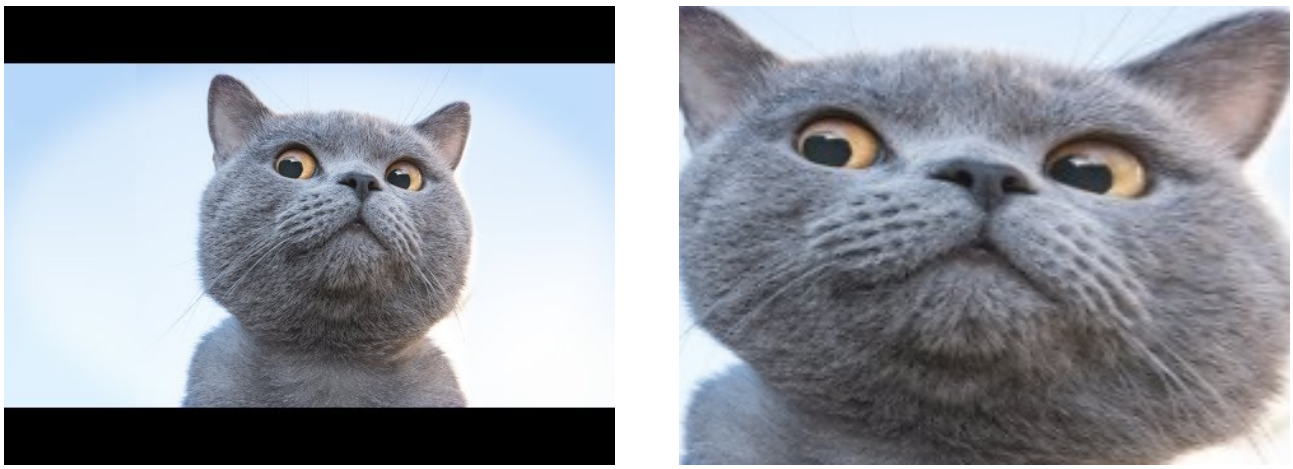

Figure 8: Projection mapping using OpenCV and C++ programming

2.2 Nonlinear Mapping

2.2.1 Projection Mapping

The new image obtained from *projection* mapping is

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} T \Rightarrow T = \begin{bmatrix} \mathrm{A} & \mathrm{B} & \mathrm{C} \\ \mathrm{D} & \mathrm{E} & \mathrm{F} \\ \mathrm{G} & \mathrm{H} & \mathrm{I} \end{bmatrix} \tag{6}$$

OpenCV code

Listing 8: Projection mapping image using OpenCV and C++ programming language

```
void lab2::projective_mapping(const cv::Mat& img, cv::Mat& new_img) {
  int new_w = 480 / 2;
  int new_h = 360 / 2;

  cv::Point2f pts1[4];
  pts1[0] = cv::Point2f(169.f, 55.f);
  pts1[1] = cv::Point2f(364.f, 74.f);
  pts1[2] = cv::Point2f(139.f, 314.f);
  pts1[3] = cv::Point2f(364.f, 314.f);
  cv::Point2f pts2[4];
  pts2[0] = cv::Point2f(0.f, 0.f);
  pts2[1] = cv::Point2f(float(new_w), 0.f);
  pts2[2] = cv::Point2f(0.f, float(new_h));
  pts2[3] = cv::Point2f(float(new_w), float(new_w));
  cv::Mat T_per = cv::getPerspectiveTransform(pts1, pts2);
  cv::Mat temp;
  cv::warpPerspective(img, new_img, T_per, cv::Size(new_w, new_h));
}
```

Result

2.2.2 Polynomial Mapping

The new image obtained from *polynomial* mapping is

$$\begin{cases} x' = a_1 + a_2 x + a_3 y + a_4 x^2 + a_5 xy + a_6 y^2 \\ y' = b_1 + b_2 x + b_3 y + b_4 x^2 + b_5 xy + b_6 y^2 \end{cases} \tag{7}$$

OpenCV code

Listing 9: Polynomial mapping image using OpenCV and C++ programming language

```
void lab2::polynomial_mapping(const Vector6d& A, const Vector6d& B,
                              const cv::Mat& img, cv::Mat& new_img) {
  for (int x = 0; x < img.cols; ++x) {
    for (int y = 0; y < img.rows; ++y) {
      int xnew, ynew;
      xnew = int(std::round(A(0) + x * A(1) + y * A(2)) + x * x * A(3) +
                 x * y * A(4) + y * y * A(5));
      ynew = int(std::round(B(0) + x * B(1) + y * B(2)) + x * x * B(3) +
                 x * y * B(4) + y * y * B(5));
      if (xnew >= 0 && xnew < img.cols && ynew >= 0 && ynew < img.rows)
        new_img.at<cv::Vec3b>(ynew, xnew) = img.at<cv::Vec3b>(y, x);
    }
  }
}
```

Result

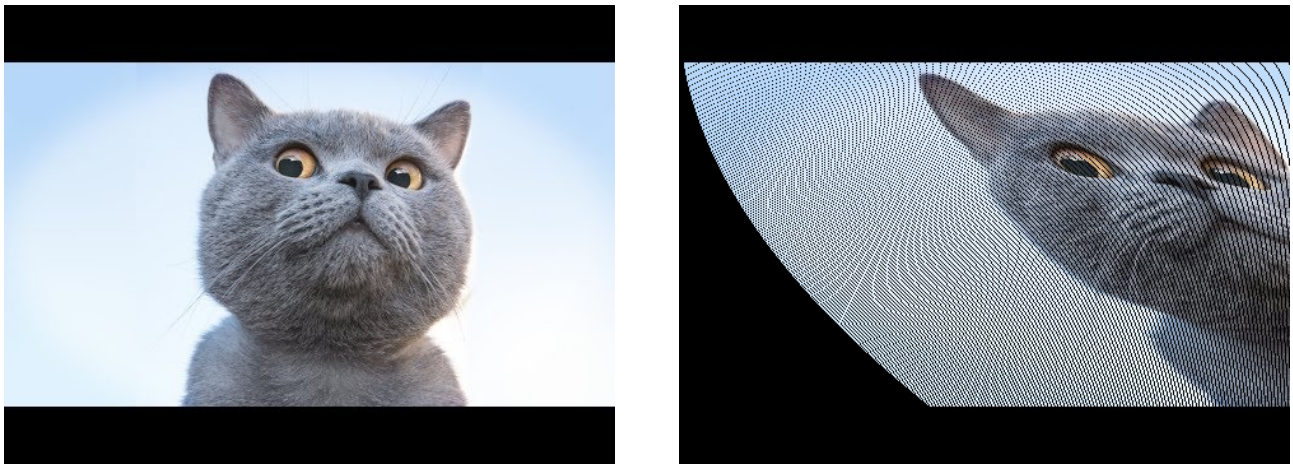

Figure 9: Polynomial mapping image using OpenCV and C++ programming

2.2.3 Sinusoidal Distortion

Harmonic distortion of an image can be considered as another the example of nonlinear transformation.

OpenCV code

Listing 10: Harmonic distortion image using OpenCV and C++ programming language

```
void lab2::sinusoidal_distortion(const cv::Mat& img, cv::Mat& new_img) {
  cv::Mat u = cv::Mat::zeros(img.rows, img.cols, CV_32F);
  cv::Mat v = cv::Mat::zeros(img.rows, img.cols, CV_32F);
  for (int x = 0; x < img.cols; ++x) {
    for (int y = 0; y < img.rows; ++y) {
      u.at<float>(y, x) = float(x + 20 * sin(2 * M_PI * y / 90));
      v.at<float>(y, x) = float(y);
    }
  }
  cv::remap(img, new_img, u, v, cv::INTER_LINEAR);
}
```

Result

Figure 10: Harmonic distortion using OpenCV and C++ programming

2.3 Distortion correction

Fisheye distortion

$$\theta_d = \theta \left(1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8\right) \tag{8}$$

The distorted point coordinates are $[x'\, y']$ where

$$\begin{aligned} x' &= (\theta_d/r)\, a \\ y' &= (\theta_d/r)\, b \end{aligned} \tag{9}$$

Finally, conversion into pixel coordinates: The final pixel coordinates $[u\ v]$ where:

$$\begin{aligned} u &= f_x\left(x' + \alpha y'\right) + c_x \\ v &= f_y y' + c_y \end{aligned} \tag{10}$$

OpenCV code

Listing 11: Distortion correction of a fisheyes with known camera parameters image using OpenCV and C++ programming language

```
void lab2::undistort_fisheye(const cv::Mat& dist_img, cv::Mat& undist_img) {

  // Camera intrinsic matrix K
  double fx = 2.8498089599609375e+02;
  double fy = 2.8610238647460938e+02;
  double cx = 4.2524438476562500e+02;
  double cy = 3.9846759033203125e+02;
  double k1 = -7.3047108016908169e-03;
  double k2 = 4.3499931693077087e-02;
  double k3 = -4.1283041238784790e-02;
  double k4 = 7.6524601317942142e-03;

  cv::Mat cameraMatrix = cv::Mat(3, 3, cv::DataType<double>::type);
  cv::Mat distortionCoeffs = cv::Mat(4, 1, cv::DataType<double>::type);

  cameraMatrix.at<double>(0, 0) = fx;
  cameraMatrix.at<double>(0, 1) = 0;
  cameraMatrix.at<double>(0, 2) = cx;
  cameraMatrix.at<double>(1, 0) = 0;
  cameraMatrix.at<double>(1, 1) = fy;
  cameraMatrix.at<double>(1, 2) = cy;
  cameraMatrix.at<double>(2, 0) = 0;
  cameraMatrix.at<double>(2, 1) = 0;
  cameraMatrix.at<double>(2, 2) = 1;

```

```
  distortionCoeffs.at<double>(0, 0) = k1;
  distortionCoeffs.at<double>(1, 0) = k2;
  distortionCoeffs.at<double>(2, 0) = k3;
  distortionCoeffs.at<double>(3, 0) = k4;

  cv::Mat E = cv::Mat::eye(3, 3, cv::DataType<double>::type);

  cv::Size size = {dist_img.cols, dist_img.rows};

  cv::Mat map1;
  cv::Mat map2;

  cv::fisheye::initUndistortRectifyMap(cameraMatrix, distortionCoeffs, E,
                                       cameraMatrix, size, CV_16SC2, map1,
                                       map2);

  cv::remap(dist_img, undist_img, map1, map2, cv::INTER_LINEAR,
            CV_HAL_BORDER_CONSTANT);
}
```

Figure 11: Distortion correction of a fisheyes with known camera parameters image using OpenCV and C++ programming language

2.4 Automatic Panorama Stitching

OpenCV library provides a special class for automatic stitching of images called Stitcher. It is designed to work either for stitching panoramic photos cv::Stitcher::PANORAMA in C++.

OpenCV code

Listing 12: Automatic panorama stitching with OpenCV of a pair image using OpenCV and C++ programming language

```
bool lab2::pano_stitcher(const std::vector<cv::Mat>& vImgs, cv::Mat& pano) {
  cv::Ptr<cv::Stitcher> stitcher = cv::Stitcher::create(cv::Stitcher::
    PANORAMA);
  cv::Stitcher::Status status = stitcher->stitch(vImgs, pano);
  return status;
}
```

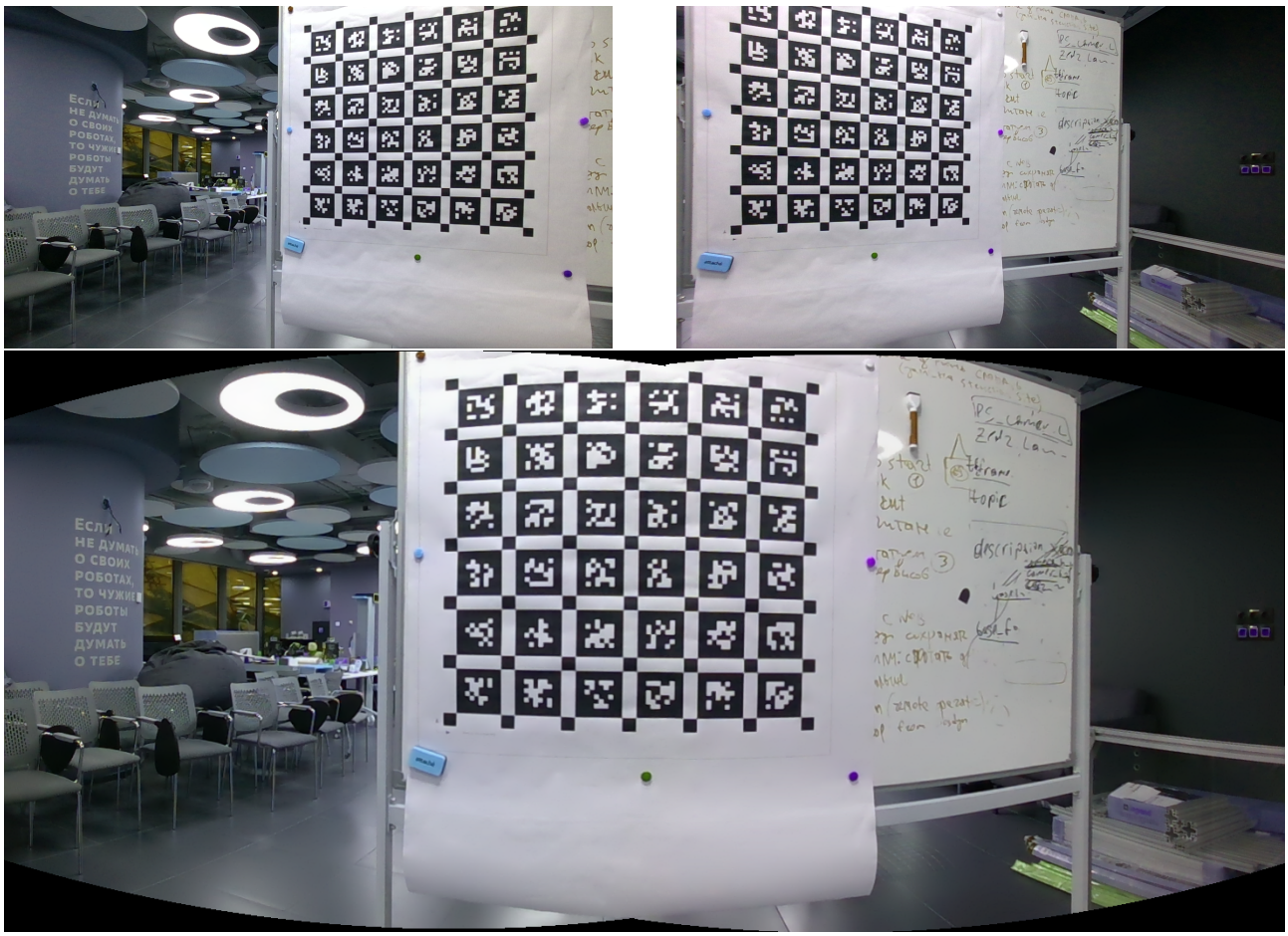

Figure 12: Automatic panorama stitching with OpenCV of a pair image using OpenCV and C++ programming language

3 Conclusion

In this work, we implement several main mapping types and using geometric transformations for spatial image correction.