



НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: ТЕХНИЧЕСКОЕ ЗРЕНИЕ

Отчет по лабораторной работе №2

“Геометрические преобразования изображений”

Выполнили:

Новиков Дмитрий, ТЕХ.ЗРЕНИЕ 1.1

Преподаватель:

Шаветов С. В.

Санкт-Петербург, 2024

Содержание

1. Теоретическая часть	2
1.1. Матрица преобразования в однородной системе координат	2
2. Линейные преобразования	3
2.1. Сдвиг изображения	4
2.2. Отражение изображения	5
2.3. Однородное масштабирование изображения	7
2.4. Поворот изображения	8
2.5. Аффинное отображение	12
2.6. Скос изображения	14
2.7. Кусочно-линейное преобразование	15
3. Нелинейные преобразования	16
3.1. Проекционное отображение	16
3.2. Полиномиальное преобразование	17
3.3. Синусоидальное искажение	19
3.4. Коррекция дисторсии	20
4. Сшивка изображений	23
5. Выводы	26
6. Ответы на вопросы	27

[Исходный код на GitHub.](#)

1. Теоретическая часть

Геометрические преобразования подразумевают пространственное изменение положения пикселей изображения, при этом интенсивность пикселей остается неизменной.

В двумерных плоских геометрических преобразованиях, как правило, используется декартова система координат. В таком случае, перемещение пикселей на изображении можно описать с помощью матрицы преобразования.

Для общности с дальнейшими преобразованиями, будем использовать *однородные координаты*.

Размерность однородного пространства равна $n - 1$, где n – количество координат. Точка в однородном пространстве определяется как набор элементов $(x_0 : x_1 : \dots : x_n)$, при этом хотя бы один из элементов не должен быть равен 0. Кроме того, две точки в проективном пространстве считаются равными, если одну можно получить умножением другой на скаляр: $(x_1 : x_2 : \dots : x_n) = (\lambda x_1 : \lambda x_2 : \dots : \lambda x_n)$. То есть разные наборы координат могут задавать одну и ту же точку.

1.1. Матрица преобразования в однородной системе координат

Матрица преобразования в такой системе координат выглядит следующим образом:

$$T = \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} \quad (1)$$

Коэффициенты $\begin{bmatrix} A & B \\ D & E \end{bmatrix}$ из этой матрицы задают стандартное преобразование плоскости в декартовой системе координат, например, поворот относительно центра координат, масштабирование плоскости, центральная симметрия. Коэффициенты $\begin{bmatrix} C \\ F \end{bmatrix}$ – коэффициенты

перемещения в направлениях x и y соответственно. Коэффициенты $[G \ H]$ отвечают за геометрическое проецирование. Последний коэффициент I отвечает за масштабирование.

Для получения преобразованного изображения необходимо умножить матрицу преобразования на матрицу координат изображения:

$$X' = T \times X \quad (2)$$



Рис. 1: Исходное изображение

2. Линейные преобразования

Линейное отображение – такое отображение, при котором сохраняется форма бесконечно малых фигур и углы между прямыми в точках их пересечения. Эти преобразования являются частным случаем *аффинных преобразований*.

2.1. Сдвиг изображения

Для того, чтобы сдвинуть изображение на вектор (dx, dy) , необходимо умножить координаты каждой точки изображения на матрицу преобразования:

$$T = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \quad (3)$$

Соответственно, преобразованное изображение I' можно получить следующим образом:

$$X' = T \times X \quad (4)$$

Рассмотрим как применить сдвиг на 150 пикселей вправо и 100 пикселей вниз к изображению:

```
T = (cv::Mat_<double>(2, 3) <<
      1, 0, 150,
      0, 1, 100);
cv::warpAffine(image, image_shift, T, cv::Size(image.cols, image.rows));
cv::imwrite(path + "/lab2/outputs/image_shift.png", image_shift);
cv::imshow("Shift image", image_shift);
```

Листинг 1: Исходный код для сдвига изображения

Для удобства отладки и сборки используется средство автоматизации сборки ПО:

```
cmake_minimum_required(VERSION 2.8)

project( CV-lab2 )
find_package( OpenCV REQUIRED )

include_directories( ${OpenCV_INCLUDE_DIRS} )
add_executable( ${PROJECT_NAME} src/lab2.cpp )

target_link_libraries( ${PROJECT_NAME} ${OpenCV_LIBS} )
```

Листинг 2: CMakeLists.txt для сборки проекта

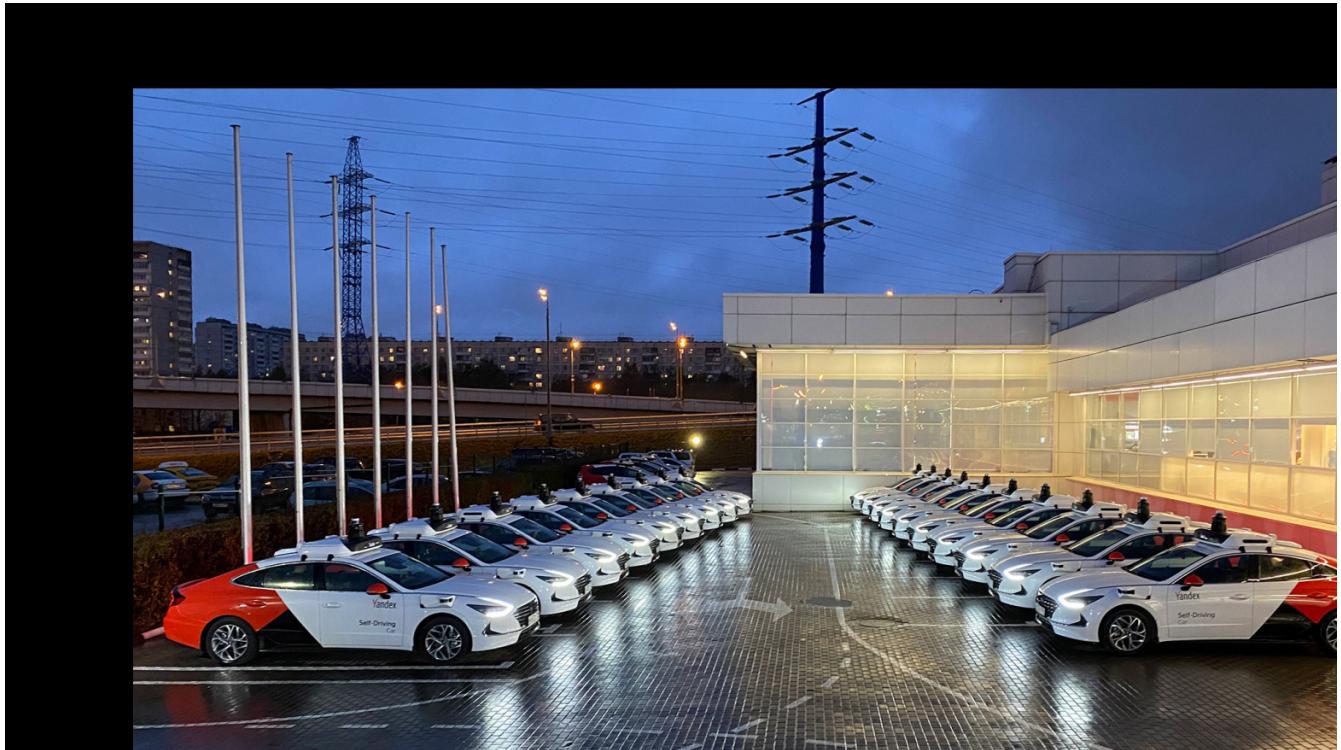


Рис. 2: Сдвиг изображения

2.2. Отражение изображения

Отражение изображения относительно оси ОХ можно получить с помощью следующей матрицы преобразования:

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & h - 1 \\ 0 & 0 & 1 \end{bmatrix} \quad (5)$$

где h – высота изображения.

Отражение изображения относительно оси ОY можно получить с помощью следующей матрицы преобразования:

$$T = \begin{bmatrix} -1 & 0 & w - 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (6)$$

где w – ширина изображения.

Использование коэффициентов $h - 1$ и $w - 1$ обусловлено тем, что нам необходимо, чтобы после отражения изображение оставалось в пределах координатной сетки.

Рассмотрим как применить отражение изображения относительно оси ОХ и ОY:

```
T = (cv::Mat<double>(2, 3) <<
```

```

    1, 0, 0,
    0, -1, image.rows-1);

cv::warpAffine(image, image_reflect, T, cv::Size(image.cols,
image.rows));

cv::imwrite(path + "/lab2/outputs/image_reflect.png", image_reflect);

cv::imshow("Shift image", image_reflect);
cv::waitKey(0);

```

Листинг 3: Исходный код для отражения изображения относительно оси ОХ



Рис. 3: Отражение изображения относительно оси ОХ

```

cv::flip(image, image_reflect, 1);

cv::imwrite(path + "/lab2/outputs/image_flip.png", image_reflect);

cv::imshow("Shift image", image_reflect);
cv::waitKey(0);

```

Листинг 4: Исходный код для отражения изображения относительно оси ОY



Рис. 4: Отражение изображения относительно оси ОY

2.3. Однородное масштабирование изображения

Для однородного масштабирования изображения на коэффициент s необходимо использовать следующую матрицу преобразования:

$$T = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (7)$$

Рассмотрим как применить однородное масштабирование изображения с коэффициентом 0.5:

```
T = (cv::Mat_<double>(2, 3) <<
      0.5, 0, 0,
      0, 0.5, 0);

cv::warpAffine(image, image_scale, T, cv::Size(int(0.5 * image.cols),
                                              int(0.5 * image.rows)));

cv::imwrite(path + "/lab2/outputs/image_scale.png", image_scale);

cv::imshow("Shift image", image_scale);
cv::waitKey(0);
```

Листинг 5: Исходный код для однородного масштабирования изображения



Рис. 5: Однородное масштабирование изображения

Кроме того, можно использовать функцию `cv::resize` из библиотеки OpenCV для однородного масштабирования изображения:

```
cv::resize(image, image_resize, cv::Size(int(0.5 * image.cols), int(0.5 * image.rows)));

cv::imwrite(path + "/lab2/outputs/image_resize.png", image_resize);

cv::imshow("Shift image", image_resize);
cv::waitKey(0);
```

Листинг 6: Исходный код для однородного масштабирования изображения с использованием библиотеки OpenCV

2.4. Поворот изображения

Для поворота изображения на угол θ необходимо использовать следующую матрицу преобразования:

$$T = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (8)$$



Рис. 6: Однородное масштабирование изображения с использованием библиотеки OpenCV

Рассмотрим как применить поворот изображения на 25 градусов:

```
double phi = 25 * M_PI / 180;
T = (cv::Mat<double>(2, 3) <<
    cos(phi), -sin(phi), 0,
    sin(phi), cos(phi), 0);

cv::warpAffine(image, image_rotate, T, cv::Size(image.cols, image.rows));

cv::imwrite(path + "/lab2/outputs/image_rotate.png", image_rotate);

cv::imshow("Shift image", image_rotate);
cv::waitKey(0);
```

Листинг 7: Исходный код для поворота изображения

Видим, что изображения действительно повернулось на 25 градусов против часовой стрелки относительно начала координат, которое находится в левом верхнем углу изображения.

Для того, чтобы осуществить поворот изображения относительно определенной точки, необходимо выполнить следующие шаги:

1. Переместить изображение так, чтобы точка, относительно которой будет осуществляться поворот, находилась в начале координат



Рис. 7: Поворот изображения

2. Повернуть изображение относительно начала координат
3. Переместить изображение обратно

Матрицы для каждого шага будут следующими:

$$T_1 = \begin{bmatrix} 1 & 0 & -\frac{w}{2} \\ 0 & 1 & -\frac{h}{2} \\ 0 & 0 & 1 \end{bmatrix} \quad (9)$$

$$T_2 = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (10)$$

$$T_3 = T_1^{-1} = \begin{bmatrix} 1 & 0 & \frac{w}{2} \\ 0 & 1 & \frac{h}{2} \\ 0 & 0 & 1 \end{bmatrix} \quad (11)$$

Применение этих матриц к изображению позволит осуществить поворот изображения относительно центра:

$$X' = T_3 \times T_2 \times T_1 \times I \quad (12)$$

Рассмотрим как применить поворот изображения на 10 градусов относительно центра:

```

double phi = 25 * M_PI / 180;

T1 = (cv::Mat_<double>(3, 3) <<
    1, 0, -(image.cols-1) / 2.0,
    0, 1, -(image.rows-1) / 2.0,
    0, 0, 1);

T2 = (cv::Mat_<double>(3, 3) <<
    cos(phi), -sin(phi), 0,
    sin(phi), cos(phi), 0,
    0, 0, 1);

T3 = (cv::Mat_<double>(3, 3) <<
    1, 0, (image.cols-1) / 2.0,
    0, 1, (image.rows-1) / 2.0,
    0, 0, 1);

T = cv::Mat(T3 * T2 * T1, cv::Rect(0, 0, 3, 2));

cv::warpAffine(image, image_rotate_point, T, cv::Size(image.cols,
    image.rows));

cv::imwrite(path + "/lab2/outputs/image_rotate_point.png",
    image_rotate_point);

cv::imshow("Shift image", image_rotate_point);
cv::waitKey(0);

```

Листинг 8: Исходный код для поворота изображения с использованием библиотеки OpenCV

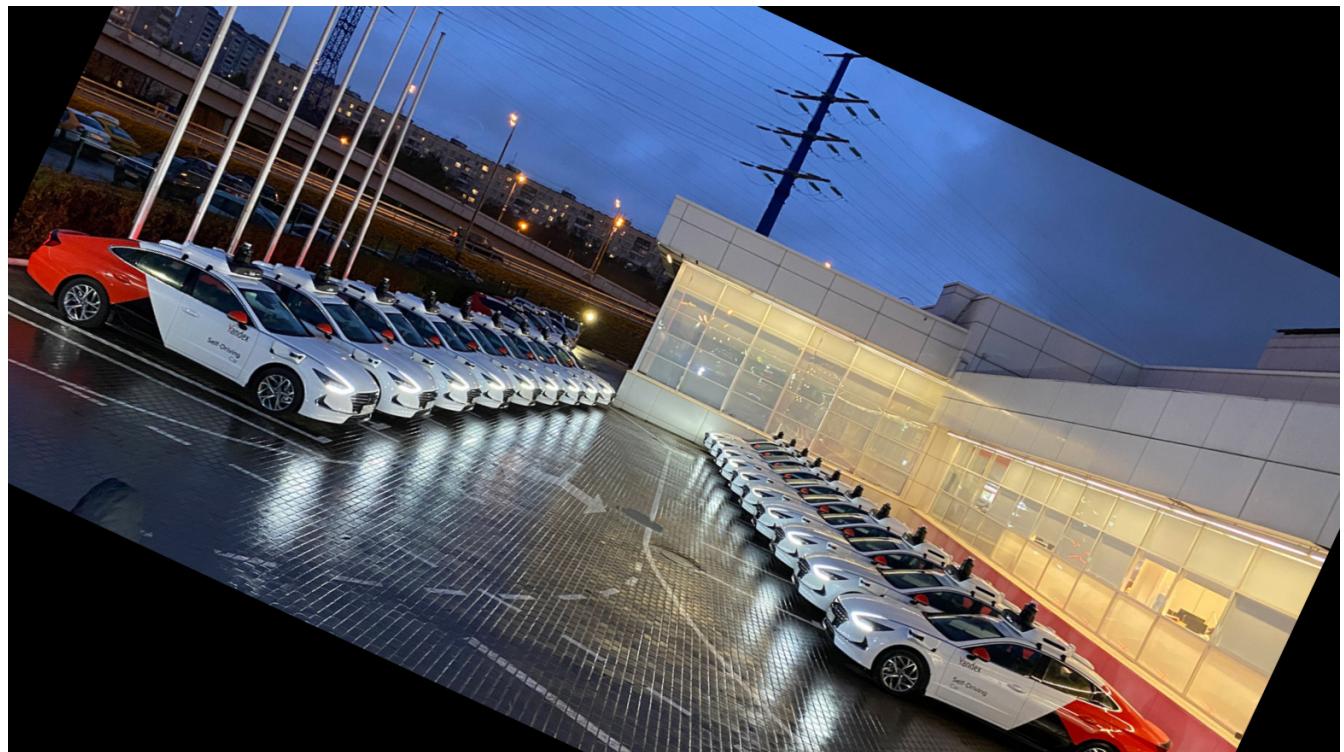


Рис. 8: Поворот изображения относительно центра

Кроме того, можно использовать функцию `cv2.getRotationMatrix2D` из библиотеки OpenCV для поворота изображения относительно центра:

```
double phi = 25;
T = cv::getRotationMatrix2D(cv::Point2d(((image.cols-1) / 2.0),
    (image.rows-1) / 2.0), -phi, 1);
cv::warpAffine(image, image_rotate_point_func, T, cv::Size(image.cols,
    image.rows));

cv::imwrite(path + "/lab2/outputs/image_rotate_point_func.png",
    image_rotate_point_func);

cv::imshow("Shift image", image_rotate_point_func);
cv::waitKey(0);
```

Листинг 9: Исходный код для поворота изображения относительно центра с использованием библиотеки OpenCV

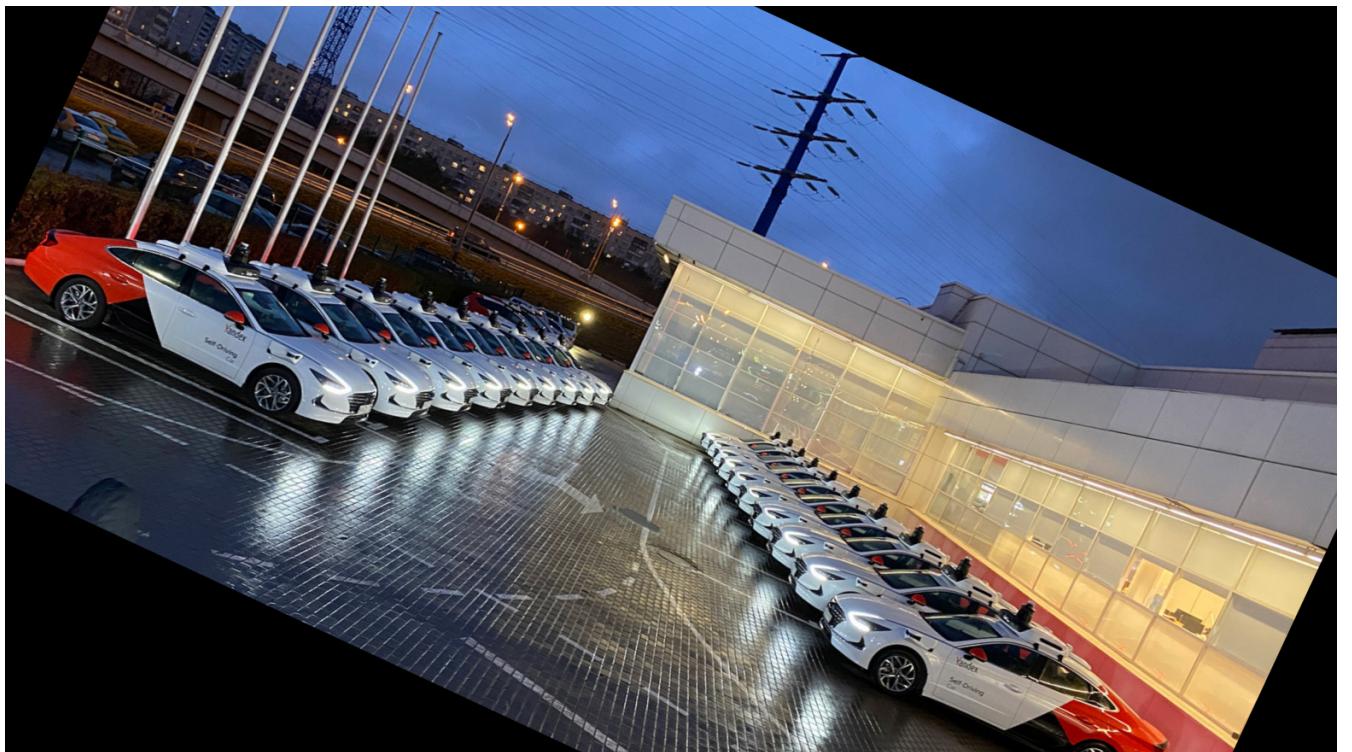


Рис. 9: Поворот изображения с использованием библиотеки OpenCV

2.5. Аффинное отображение

Аффинное преобразование – это преобразование, которое сохраняет параллельность прямых, соотношение длин отрезков, лежащих на одной прямой, и углы между пересекающимися прямыми. Аффинные преобразования являются подмножеством проективных преобразований.

Для задания аффинного преобразования можно использовать три пары соответствующих точек на исходном и преобразованном изображениях.

Программная реализация аффинного преобразования:

```
std::vector<cv::Point2f> pts_src = {{50, 300},  
                                      {150, 200},  
                                      {50, 50}};  
std::vector<cv::Point2f> pts_dst = {{50, 310},  
                                      {160, 200},  
                                      {50, 60}};  
  
T = cv::getAffineTransform(pts_src, pts_dst);  
  
cv::warpAffine(image, image_Affine, T, cv::Size(image.cols, image.rows));  
  
cv::imwrite(path + "/lab2/outputs/image_Affine.png", image_Affine);  
  
cv::imshow("Shift image", image_Affine);  
cv::waitKey(0);
```

Листинг 10: Исходный код для скоса изображения



Рис. 10: Аффинное отображение

2.6. Скос изображения

Для того, чтобы осуществить скос изображения, необходимо использовать следующую матрицу преобразования:

$$T_1 = \begin{bmatrix} 1 & \alpha & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ или } T_2 = \begin{bmatrix} 1 & 0 & 0 \\ \beta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (13)$$

Рассмотрим как применить скос изображения:

```
double s = 0.2;
T = (cv::Mat<double>(2, 3) <<
      1, s, 0,
      0, 1, 0);

cv::warpAffine(image, image_bevel, T, cv::Size(image.cols, image.rows));

cv::imwrite(path + "/lab2/outputs/image_bevel.png", image_bevel);

cv::imshow("Shift image", image_bevel);
cv::waitKey(0);
```

Листинг 11: Исходный код для скоса изображения



Рис. 11: Скос изображения

2.7. Кусочно-линейное преобразование

Кусочно-линейное преобразование – это преобразование, при котором изображение разбивает на части, и каждая часть преобразуется отдельно. Например, можно рассмотреть преобразование, которое левую часть оставляет без изменений, а правую часть растягивает в 4 раза.

```
double stretch = 2;
T = (cv::Mat_<double>(2, 3) <<
      stretch, 0, 0,
      0, 1, 0);

image_clone = image.clone();
image_piece_r = image_clone(cv::Rect(image.cols / 2, 0, image.cols / 2,
                                       image.rows));

cv::warpAffine(image_piece_r, image_piece_r, T,
               cv::Size(image_piece_r.cols, image.rows));

cv::imwrite(path + "/lab2/outputs/image_piece_r.png", image_clone);

cv::imshow("Shift image", image_clone);
cv::waitKey(0);
```

Листинг 12: Исходный код для кусочно-линейного преобразования



Рис. 12: Кусочно-линейное преобразование

3. Нелинейные преобразования

В реальности необходимо применять не только линейные преобразования, но и нелинейные. Например, для коррекции дисторсии, которая вызывается несовершенством оптики, необходимо использовать нелинейные преобразования.

3.1. Проекционное отображение

Проекционное отображение – это отображение, которое оставляет прямые линии прямыми, но геометрия изображения может быть искажена, так как данное отображение не сохраняет углы между прямыми.

Матрица для проекционного отображения в общем виде выглядит следующим образом:

$$T = \begin{bmatrix} A & B & E \\ C & D & F \\ G & H & K \end{bmatrix} \quad (14)$$

где A, B, C, D – коэффициенты, которые определяют проекционное отображение, E, F – коэффициенты сдвига, G, H – коэффициенты, которые определяют проекционный масштаб, K – коэффициент масштабирования.

Рассмотрим как применить проективное отображение:

```
T = (cv::Mat<double>(3, 3) <<
      0.7, 0.3, 0.00045,
      0.2, 0.5, 0.0005,
      0, 0, 1);

cv::warpPerspective(image, image_projective, T, cv::Size(image.cols,
    image.rows));

cv::imwrite(path + "/lab2/outputs/image_projective.png", image_projective);

cv::imshow("Shift image", image_projective);
cv::waitKey(0);
```

Листинг 13: Исходный код для проективного отображения

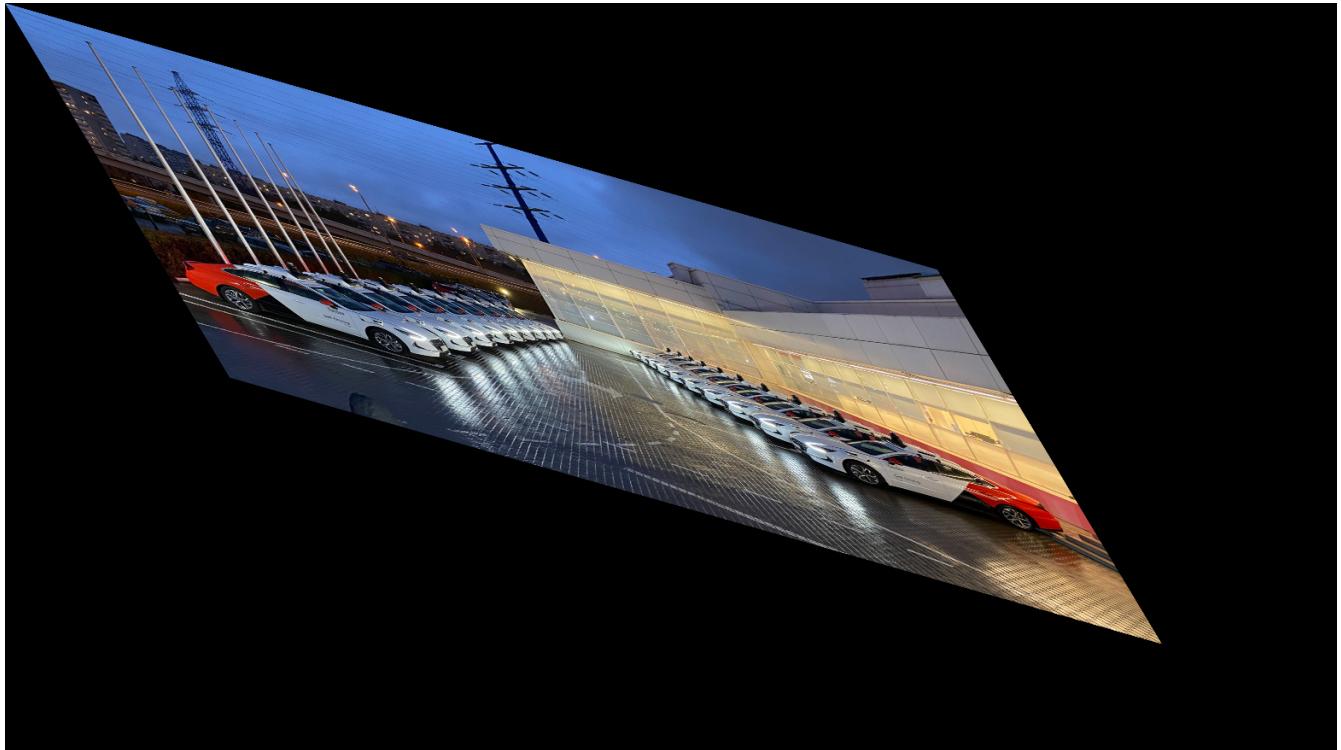


Рис. 13: Проективное отображение

3.2. Полиномиальное преобразование

Полиномиальное преобразование – это отображение исходного изображения на преобразованное с использованием полиномиальной функции.

Полиномиальное преобразование второго порядка выглядит следующим образом:

$$\begin{cases} x' = a_0 + a_1x + a_2y + a_3x^2 + a_4xy + a_5y^2 \\ y' = b_0 + b_1x + b_2y + b_3x^2 + b_4xy + b_5y^2 \end{cases} \quad (15)$$

```

const double coeff[2][6] = {{0, 1, 0, 0.00001, 0.0001, 0},
                           {0, 0, 1, 0, 0.0002, 0};

if(image.depth() == CV_8U)
    image.convertTo(image_polynomial, CV_32F, 1.0 / 255);
else
    image_polynomial = image;

std::vector<cv::Mat> image_BGR;
cv::split(image_polynomial, image_BGR);
cv::Mat i_pol;
for(int k = 0; k < image_BGR.size(); k++){

    i_pol = cv::Mat::zeros(image_BGR[k].rows,

```

```

        image_BGR[k].cols,
        image_BGR[k].type()));

for(int x = 0; x < image_BGR[k].cols; ++x){
    for(int y = 0; y < image_BGR[k].rows; ++y){

        int x_new = static_cast<int>(round(coeff[0][0] +
            x * coeff[0][1] + y * coeff[0][2] +
            x * x * coeff[0][3] + y * x * coeff[0][4] +
            y * y * coeff[0][5]));

        int y_new = static_cast<int>(round(coeff[1][0] +
            x * coeff[1][1] + y * coeff[1][2] +
            x * x * coeff[1][3] + y * x * coeff[1][4] +
            y * y * coeff[1][5]));

        if(x_new >= 0 && x_new < image_BGR[k].cols
            && y_new >= 0 && y_new < image_BGR[k].rows)
            i_pol.at<float>(y_new, x_new) = image_BGR[k].at<float>(y,
            x);
    }
}
image_BGR[k] = i_pol;
}
cv::merge(image_BGR, i_pol);

if(image.depth() == CV_8U)
    image_polynomial.convertTo(image_polynomial, CV_8UC3, 255);

cv::imwrite(path + "/lab2/outputs/image_polynomial.png", image_polynomial);
cv::imshow("B", image_polynomial);
cv::waitKey(0);

```

Листинг 14: Исходный код для полиномиального преобразования

Видим, что на изображении появились артефакты, так как не все пиксели были заполнены. Это связано с тем, что при применении полиномиального преобразования цвета пикселей могут быть не определены для некоторых координат, и в таком случае необходимо использовать интерполяцию.

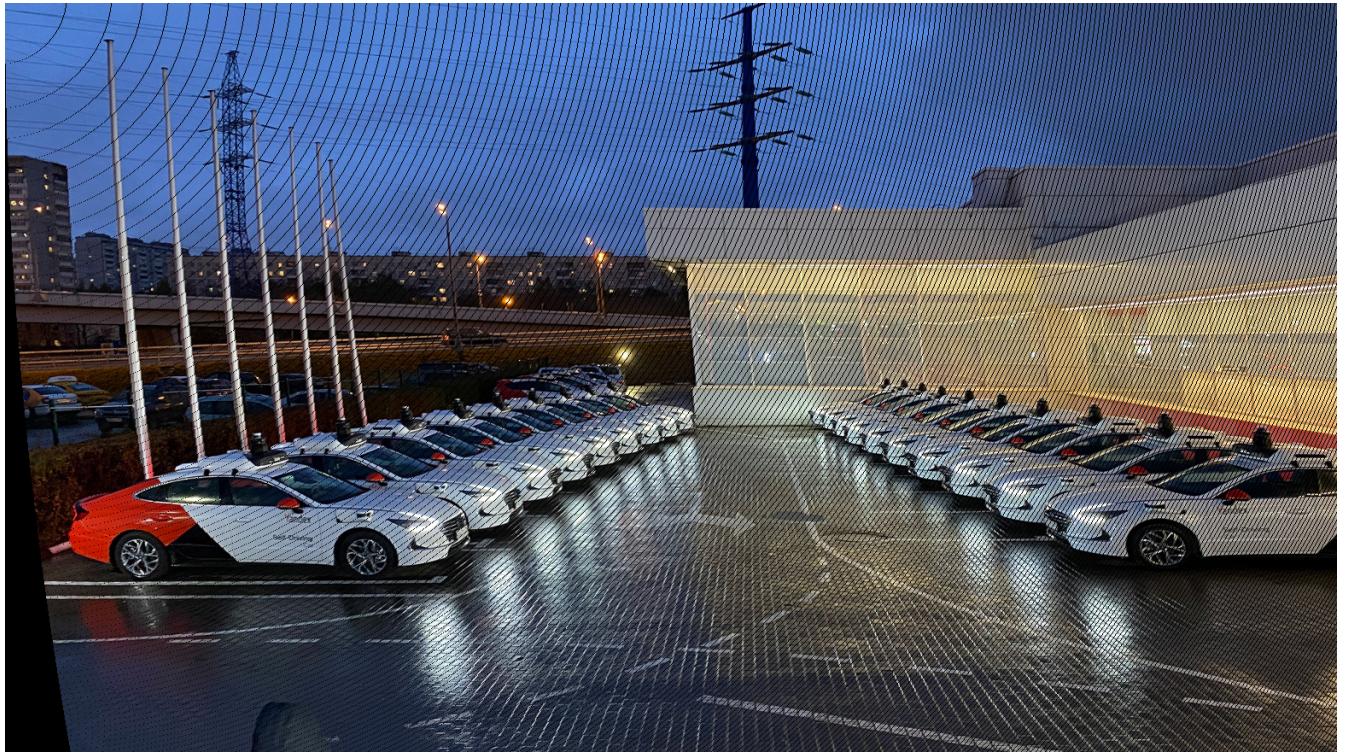


Рис. 14: Полиномиальное преобразование

3.3. Синусоидальное искажение

Еще одним примером нелинейного преобразования может быть искажение изображения по гармоническому закону.

```
cv::Mat u = cv::Mat::zeros(image.rows, image.cols, CV_32F);
cv::Mat v = cv::Mat::zeros(image.rows, image.cols, CV_32F);

for(int x = 0; x < image.cols; ++x){
    for(int y = 0; y < image.rows; ++y){
        u.at<float>(y, x) = float(x + 20 * sin(2 * M_PI * y / 90));
        v.at<float>(y, x) = float(y);
    }
}
cv::Mat image_sinusoid;
cv::remap(image, image_sinusoid, u, v, cv::INTER_LINEAR);

cv::imwrite(path + "/lab2/outputs/image_sinusoid.png", image_sinusoid);
cv::imshow("B", image_sinusoid);
cv::waitKey(0);
```

Листинг 15: Исходный код для синусоидального искажения

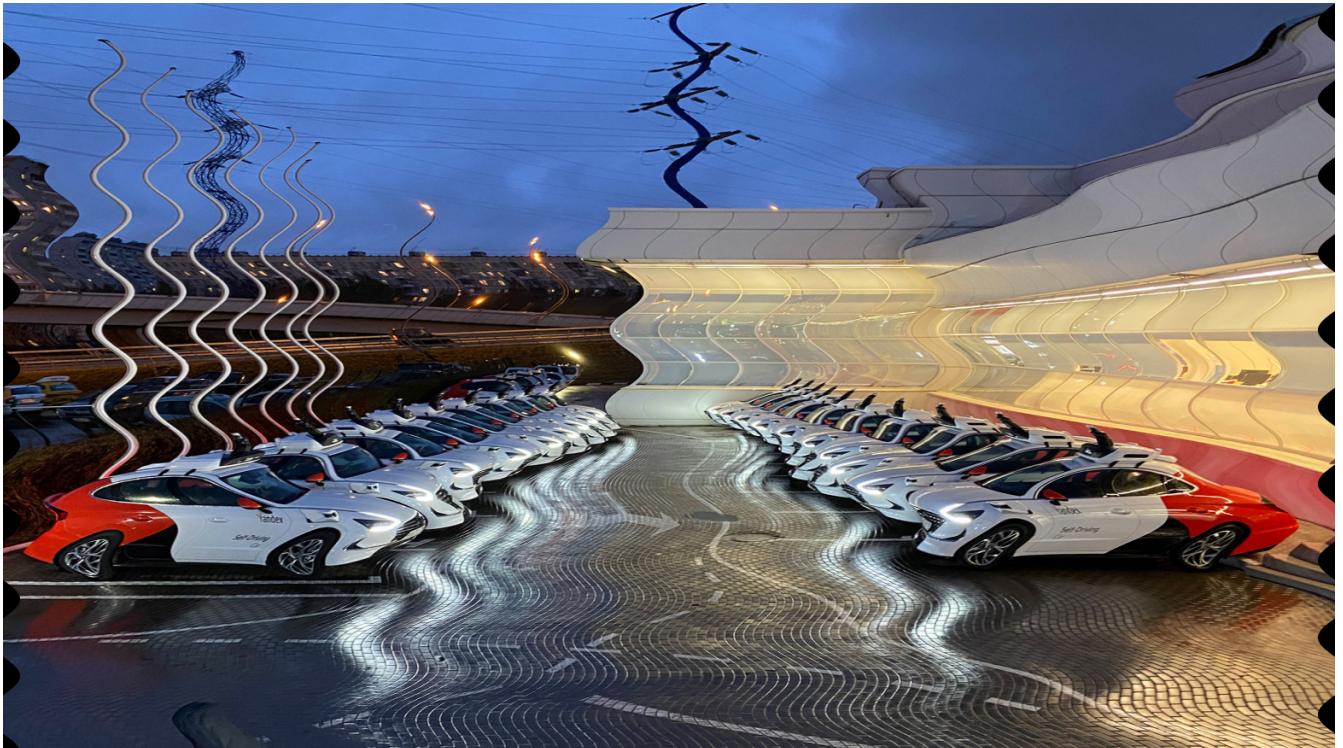


Рис. 15: Синусоидальное искажение

3.4. Коррекция дисторсии

При формировании изображения оптической системой камеры, на нем может возникнуть дисторсия, которая вызвана несовершенством оптики. **Дисторсия** – это оптическое искажение, которое искривляет прямые линии на изображении.

```
cv::Mat x_i, y_i;
std::vector<float> t_x, t_y;

for(int i = 0; i < image.cols; ++i)
    t_x.push_back(float(i));

for(int i = 0; i < image.rows; ++i)
    t_y.push_back(float(i));

cv::repeat(cv::Mat(t_x).reshape(1, 1), image.rows, 1, x_i);
cv::repeat(cv::Mat(t_y).reshape(1, 1).t(), 1, image.cols, y_i);

double x_mid = x_i.cols / 2.0;
double y_mid = x_i.rows / 2.0;

x_i -= x_mid;
x_i /= x_mid;
y_i -= y_mid;
y_i /= y_mid;

cv::Mat r, theta;
```

```

cv::cartToPolar(x_i, y_i, r, theta);

double F3 = 0.1, F5 = 0.1;
cv::Mat r3, r5;

pow(r, 3, r3); pow(r, 5, r5);
r += r3 * F3; r += r5 * F5;

cv::Mat u, v;
cv::polarToCart(r, theta, u, v);
u *= x_mid;
u += x_mid;
v *= y_mid;
v *= y_mid;

cv::Mat image_barrel;
cv::remap(image, image_barrel, u, v, cv::INTER_LINEAR);

cv::imwrite(path + "/lab2/outputs/image_barrel.png", image_barrel);
cv::imshow("B", image_barrel);
cv::waitKey(0);

```

Листинг 16: Исходный код для коррекции бочкообразной дисторсии



Рис. 16: Коррекция дисторсии (бочка)

```

cv::Mat x_i, y_i;
std::vector<float> t_x, t_y;

for(int i = 0; i < image.cols; ++i)
    t_x.push_back(float(i));

```

```

for(int i = 0; i < image.rows; ++i)
    t_y.push_back(float(i));

cv::repeat(cv::Mat(t_x).reshape(1, 1), image.rows, 1, x_i);
cv::repeat(cv::Mat(t_y).reshape(1, 1).t(), 1, image.cols, y_i);

double x_mid = x_i.cols / 2.0;
double y_mid = x_i.rows / 2.0;

x_i -= x_mid;
x_i /= x_mid;
y_i -= y_mid;
y_i /= y_mid;

cv::Mat r, theta;
cv::cartToPolar(x_i, y_i, r, theta);

double F3 = -0.3;
cv::Mat r3, r5;

pow(r, 3, r3); pow(r, 5, r5);
r += r + r3 * F3;

cv::Mat u, v;
cv::polarToCart(r, theta, u, v);
u *= x_mid;
u += x_mid;
v *= y_mid;
v *= y_mid;

cv::Mat image_barrel;
cv::remap(image, image_barrel, u, v, cv::INTER_LINEAR);

cv::imwrite(path + "/lab2/outputs/image_pincushion.png", image_barrel);
cv::imshow("B", image_barrel);
cv::waitKey(0);

```

Листинг 17: Исходный код для коррекции подушкообразной дисторсии



Рис. 17: Коррекция дисторсии (подушка)

4. Сшивка изображений

Сшивка изображений – это процесс объединения двух или более изображений в одно. Для того, чтобы осуществить сшивку изображений, необходимо найти общие точки на изображениях, и на основе этих точек осуществить сшивку, перейдя от системы координат одного изображения к системе координат другого изображения.

Посмотрим на два изображения, которые необходимо сплить:

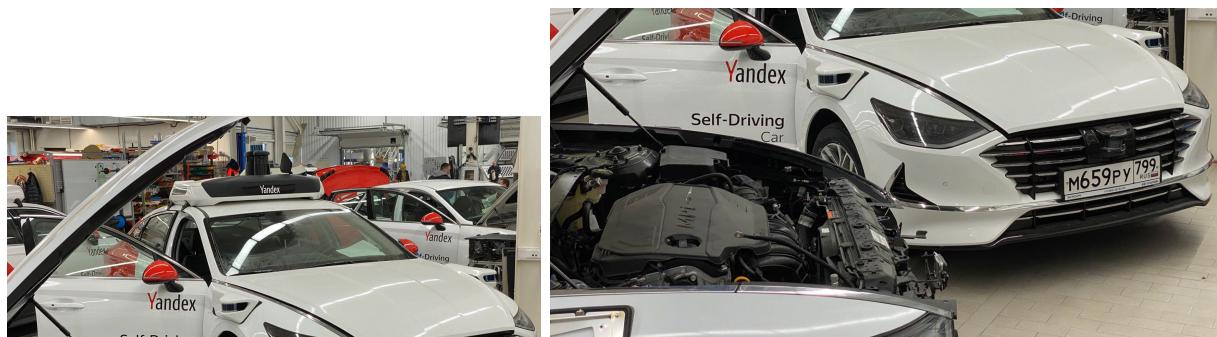


Рис. 18: Изображения для сшивки

На рисунке 18 видно, что изображения имеют общую часть, и на основе этой части можно осуществить сшивку.

Посмотрим на несколько способов сшивки изображений:

```
cv::Mat topPart = cv::imread(path + "/lab2/1.jpg");
cv::Mat botPart = cv::imread(path + "/lab2/2.jpg");

int templ_size = 10;
cv::Mat templ = topPart(cv::Rect(0,topPart.rows - templ_size - 1,
                                  topPart.cols, templ_size));

cv::Mat res;
cv::matchTemplate(botPart, templ, res, cv::TM_CCOEFF);

double min_val, max_val;
cv::Point2i min_loc, max_loc;
cv::minMaxLoc(res, &min_val, &max_val, &min_loc, &max_loc);

cv::Mat image_res = cv::Mat::zeros(topPart.rows + botPart.rows -
                                    max_loc.y - templ_size, topPart.cols, topPart.type());

topPart.copyTo(image_res(cv::Rect(0, 0, topPart.cols, topPart.rows)));

botPart(cv::Rect(0, max_loc.y + templ_size, botPart.cols,
                 botPart.rows - max_loc.y - templ_size)).
    copyTo(image_res(cv::Rect(0, topPart.rows,
                           botPart.cols, botPart.rows - max_loc.y - templ_size)));

cv::imwrite(path + "/lab2/outputs/image_glues.png", image_res);
cv::imshow("B", image_res);
cv::waitKey(0);
```

Листинг 18: Исходный код для сшивки изображений



Рис. 19: Сшивка изображений

Изображение выше имеет выраженную границу сшивки из-за недостаточного исследования матрицы корреляция, что послужило частично неточному определению смежных точек

```
cv::Mat topPart = cv::imread(path + "/lab2/source/top.png");
cv::Mat botPart = cv::imread(path + "/lab2/source/bttm.png");
image = cv::imread(path + "/lab2/source/image_cut.png");

cv::Ptr<cv::Stitcher> stitcher = cv::Stitcher::create(cv::Stitcher::SCANS);

std::vector<cv::Mat> imgs;
imgs.push_back(botPart);
imgs.push_back(topPart);

cv::Mat image_stitch;
cv::Stitcher::Status status = stitcher->stitch(imgs, image_stitch);

cv::resize(image_stitch, image_stitch, cv::Size(image.cols, image.rows));

cv::imwrite(path + "/lab2/outputs/image_stitch.png", image_stitch);
cv::imshow("B", image_stitch);
cv::waitKey(0);
```

Листинг 19: Исходный код для сшивки изображений с использованием библиотеки OpenCV



Рис. 20: Сшивка изображений с использованием библиотеки OpenCV

Видим, что при использовании каждого из методов изображения были спиты.

5. Выводы

В процессе выполнения лабораторной работы мы освоили основные виды отображений и использовали различные геометрические преобразования, попробовали скорректировать оптические искажения.

Также смогли оценить качество полученных изображений и убедиться, что геометрические преобразования позволяют улучшать их визуальное восприятие

6. Ответы на вопросы

Q1. Каким образом можно выполнить поворот изображения, не используя матрицу поворота?

A1. Можно воспользоваться функцией `cv::rotate()` в библиотеке OpenCV, которая используется как раз для поворота изображения. Она принимает угол вращения в градусах в качестве аргумента. По умолчанию размер готового изображения равен размеру исходного изображения, и части повернутого изображения, которые выходят за пределы исходного размера, отсекаются. Если мы хотим, чтобы повернутое изображение полностью удовлетворяло нашим требованиям, мы можем установить параметр `expand` в `True`.

Q2. Какое минимальное количество соответствующих пар точек необходимо задать на исходном и искаженном изображениях, если порядок преобразования $n = 4$?

A2. Число минимально необходимых пар точек вычисляется по формуле:

$n_{pair_{min}} = ((n + 1) \cdot (n + 2))/2$, где n - порядок преобразования. Следовательно, при $n = 4$ нам потребуется минимум 15 пар точек.

Q3. После геометрического преобразования изображения могут появиться пиксели с неопределенными значениями интенсивности. С чем это связано и как решается данная проблема?

A3. При геометрическом преобразовании, когда выполняется трансформация (поворот, масштабирование и т.д.) изображения, пиксели перемещаются и могут попасть в пространство между исходными пикселями. Это приводит к неопределенным значениям интенсивности. Для решения этой проблемы используется *интерполяция*. Она позволяет вычислить значения для неопределенных пикселей на основе соседних известных пикселей.