

ФАКУЛЬТЕТ СИСТЕМ УПРАВЛЕНИЯ И РОБОТОТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №2  
ПО ДИСЦИПЛИНЕ «ТЕХНИЧЕСКОЕ ЗРЕНИЕ»

Выполнил:

Гридусов Денис

Преподаватель:

Шаветов С.В.

Санкт-Петербург  
2024 г.

# Содержание

<b>1 Введение</b>	<b>2</b>
<b>2 Простейшие геометрические преобразования</b>	<b>3</b>
2.1 Сдвиг изображения . . . . .	3
2.2 Отражение изображения . . . . .	4
2.3 Однородное масштабирование изображения . . . . .	5
2.4 Поворот изображения . . . . .	5
2.5 Поворот вокруг центра изображения . . . . .	7
2.6 Аффинное отображение . . . . .	7
2.7 Скос изображения . . . . .	8
2.8 Кусочно-линейные преобразование . . . . .	10
2.9 Проекционное отображение . . . . .	11
2.10 Полиномиальное отображение . . . . .	11
2.11 Синусоидальное искажение . . . . .	13
<b>3 Коррекция дисторсии</b>	<b>14</b>
<b>4 «Склейка» изображений</b>	<b>20</b>
4.1 Ручная склейка изображений . . . . .	20
4.2 Автоматическая склейка в OpenCV . . . . .	22
<b>5 Заключение</b>	<b>23</b>

# 1 Введение

**Цель работы:** освоить основные виды отображений и использовать геометрические преобразования для решения задач пространственной коррекции изображений.

Код: [github](#)

## 2 Простейшие геометрические преобразования

Возьмем произвольное изображение, например:



Рис. 1: Исходное изображение

### 2.1 Сдвиг изображения

Для разминки применим самое очевидное геометрическое преобразование - сдвиг изображения. Матрица отображения  $T$  выглядит следующим образом:

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ C & F & 1 \end{bmatrix},$$

где  $F$  - величина сдвига по оси  $Oy$ , а  $C$  - величина сдвига по оси  $Ox$ .

**Листинг 2.1** Сдвиг изображения

---

```
cv::Mat img_shift(cv::Mat img, int x_hist, int y_hist) {
    cv::Mat T = (cv::Mat_ <double>(2, 3) << 1, 0, x_hist, 0, 1, y_hist);
    cv::Mat img_shift;
    cv::warpAffine(img, img_shift, T, cv::Size(img.cols, img.rows));
    return img_shift;
}
```

---

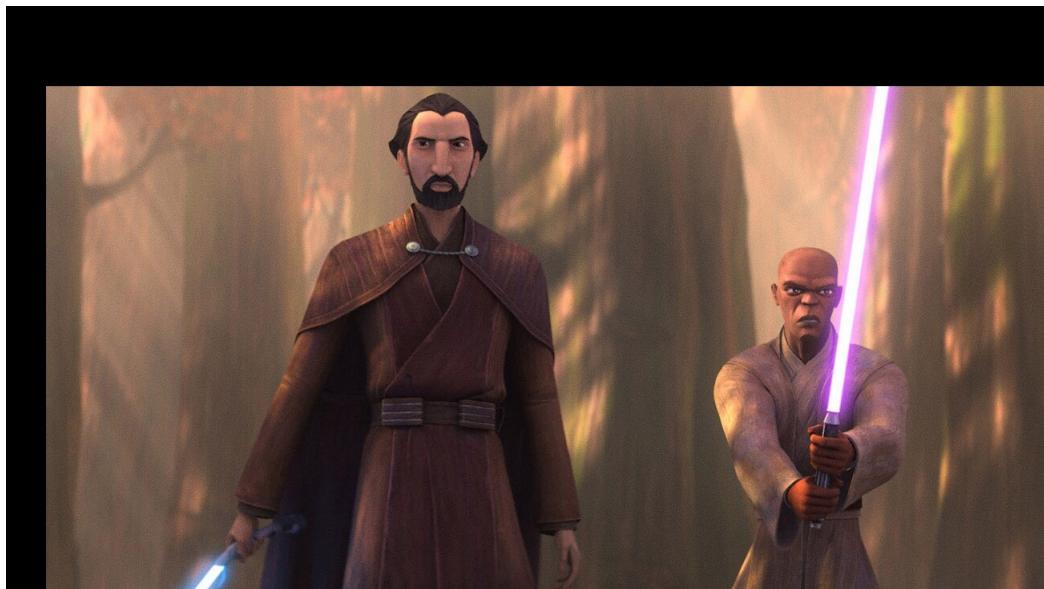


Рис. 2: Сдвиг изображения на 50 пикселей по оси X, 100 по Y

## 2.2 Отражение изображения

Отразим изображение, относительно оси O<sub>x</sub>, матрица преобразования:

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

**Листинг 2.2** Отражение относительно оси *Ox*

---

```
cv::Mat img_reflection(cv::Mat img) {
    cv::Mat T = (cv::Mat_<double>(2, 3) << 1, 0, 0, 0, 0, -1, img.rows - 1);
    cv::Mat img_reflected;
    cv::warpAffine(img, img_reflected, T, cv::Size(img.cols, img.rows));
    return img_reflected;
}
```

---

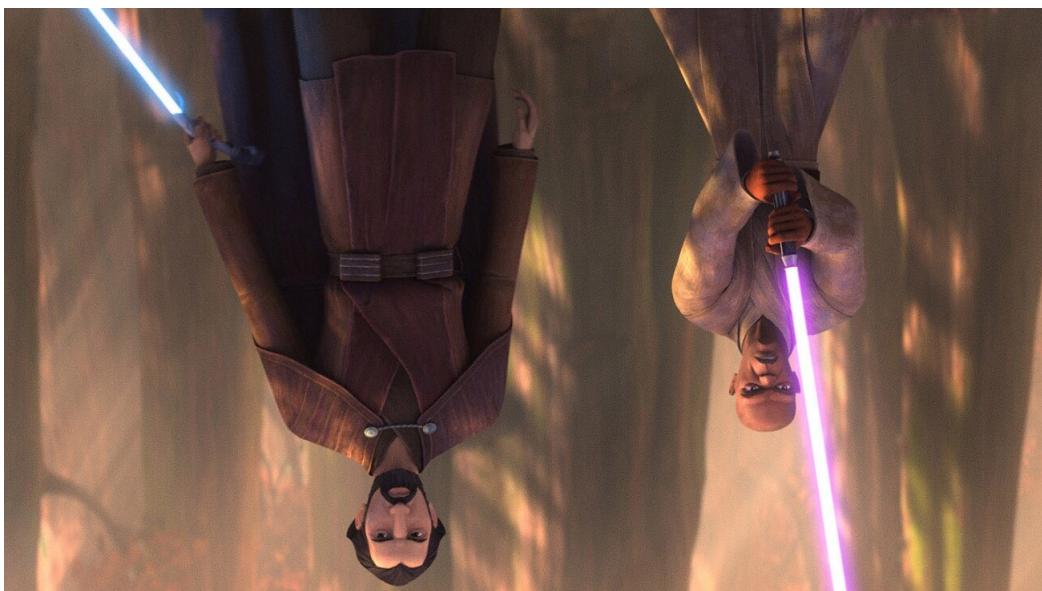


Рис. 3: Отраженное относительно оси  $Ox$  изображение

### 2.3 Однородное масштабирование изображения

А теперь масштабируем изображение, матрица преобразования:

$$T = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

где  $\alpha$  отвечает за расстяжение по оси X, а  $\beta$  - по оси Y.

**Листинг 2.3** Однородное ( $\alpha = \beta$ ) растяжение изображения

---

```
cv::Mat uniform_resize(cv::Mat img, double coeff) {
    cv::Mat T = (cv::Mat_ <double>(2, 3) << coeff, 0, 0, 0, coeff, 0);
    cv::Mat img_resized;
    cv::warpAffine(img, img_resized, T, cv::Size(int(img.cols * coeff), int(img.rows * coeff)));
    return img_resized;
}
```

---

### 2.4 Поворот изображения

Повернем изображение на произвольный угол  $\varphi$ , матрица преобразования:

$$T = \begin{bmatrix} \cos \varphi & \sin \varphi & 0 \\ -\sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

**Листинг 2.4** Поворот на произвольный угол  $\varphi$

---

```
cv::Mat img_rotate(cv::Mat img, double phi) {
    double r_phi = phi * M_PI / 180.0;
    cv::Mat T = (cv::Mat_ <double>(2, 3) << std::cos(r_phi), -std::sin(r_phi), 0,
                 std::sin(r_phi), std::cos(r_phi), 0);
```

---

```
cv::Mat img_rotated;
cv::warpAffine(img, img_rotated, T, cv::Size(img.cols, img.rows));
return img_rotated;
}
```

---



Рис. 4: Поворот на 14 градусов вокруг левого верхнего угла

## 2.5 Поворот вокруг центра изображения

Попробуем теперь повернуть изображение вокруг его центра, для этого нужно лишь сдвинуть центр в левый верхний угол, повернуть как в предыдущем пункте и потом сдвинуть центр на прежнее место. Матрица преобразования  $T$  выглядеть будет так:

$$T = \begin{bmatrix} 1 & 0 & -\frac{img.cols-1}{2} \\ 0 & 1 & -\frac{img.rows-1}{2} \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \cos \varphi & \sin \varphi & 0 \\ -\sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & \frac{img.cols-1}{2} \\ 0 & 1 & \frac{img.rows-1}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

Листинг 2.5 Поворот вокруг центра на произвольный угол  $\varphi$

```
cv::Mat img_centered_rotate(cv::Mat img, double phi) {
    double r_phi = phi * M_PI / 180.0;
    cv::Mat T2 = (cv::Mat_ <double>(3, 3) << std::cos(r_phi), -std::sin(r_phi), 0,
                  std::sin(r_phi), std::cos(r_phi), 0, 0, 0, 1);
    cv::Mat T1 = (cv::Mat_ <double>(3, 3) << 1, 0, -(img.rows - 1) / 2.0, 0, 1, -(img.cols - 1) /
                  2.0, 0, 0, 1);
    cv::Mat T3 = (cv::Mat_ <double>(3, 3) << 1, 0, (img.rows - 1) / 2.0, 0, 1, (img.cols - 1) /
                  2.0, 0, 0, 1);

    cv::Mat T = cv::Mat(T3 * T2 * T1, cv::Rect(0, 0, 3, 2));
    cv::Mat img_rotated;
    cv::warpAffine(img, img_rotated, T, cv::Size(img.cols, img.rows));
    return img_rotated;
}
```



Рис. 5: Поворот на 30 градусов вокруг центра изображения

## 2.6 Аффинное отображение

Аффинное отображение — это отображение, при котором параллельные прямые переходят в параллельные прямые, пересекающиеся в пересекающиеся, скрещивающиеся в скрещивающиеся; сохраняют-

ся отношения длин отрезков, лежащих на одной прямой (или на параллельных прямых), и отношения площадей фигур.

### Листинг 2.6 Аффинное преобразование

```
cv::Mat affine_resize(cv::Mat img) {
    std::vector<cv::Point2f> ptr_src = { {50, 100}, {350, 100}, {50, 400} };
    std::vector<cv::Point2f> ptr_dst = { {50, 150}, {200, 150}, {100, 400} };

    cv::Mat T = cv::getAffineTransform(ptr_src, ptr_dst);
    cv::Mat img_resized;
    cv::warpAffine(img, img_resized, T, cv::Size(img.cols, img.rows));

    return img_resized;
}
```

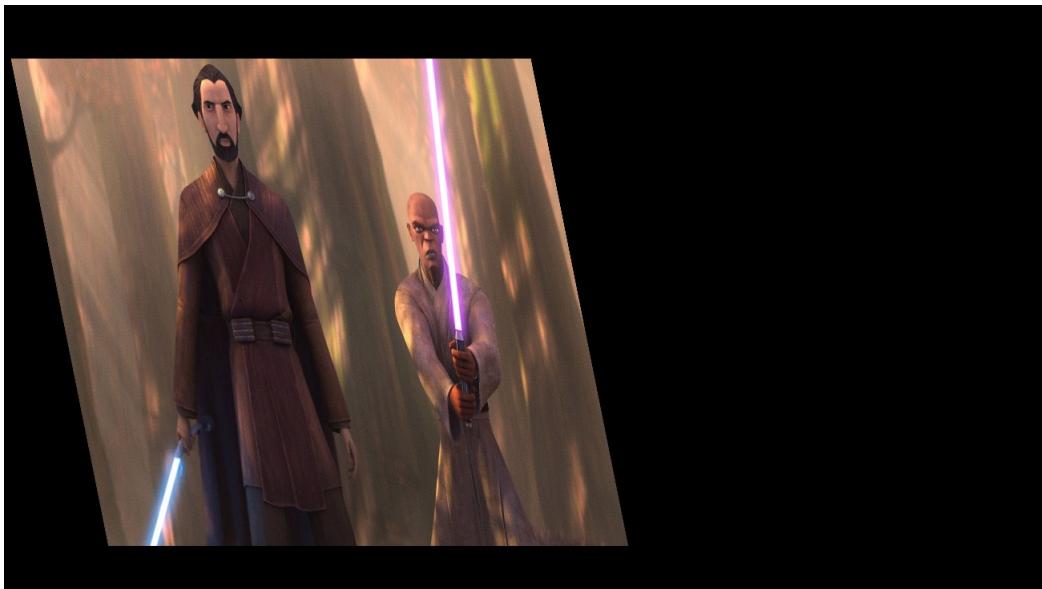


Рис. 6: Аффинное преобразование изображения

### 2.7 Скос изображения

Запишем матрицу отображения:

$$T = \begin{bmatrix} 1 & 0 & 0 \\ s & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

### Листинг 2.7 Скос изображения

```
cv::Mat img_bevel(cv::Mat img, double s) {
    cv::Mat T = (cv::Mat_<double>(2, 3) << 1, s, 0, 0, 1, 0);
    cv::Mat img_bevel;
    cv::warpAffine(img, img_bevel, T, cv::Size(img.cols, img.rows));
```

```
    return img_bevel;  
}
```

---



Рис. 7: Скос изображения с  $s = 0.3$

## 2.8 Кусочно-линейные преобразование

Теперь применим преобразование не ко всему изображению, а только к его части (к ROI).

**Листинг 2.8** Растижение только правой части изображения в 2 раз

```
cv::Mat uniform_ROI_transform(cv::Mat img, double s) {
    cv::Mat T = (cv::Mat_ <double>(2, 3) << s, 0, 0, 0, 0, 1, 0);
    cv::Mat img_piece_linear = img.clone();
    cv::Mat img_right = img_piece_linear(cv::Rect(int(img.cols / 2.0), 0, img.cols - int(img.cols / 2.0), img.rows));
    cv::warpAffine(img_right, img_right, T, cv::Size(img.cols - int(img.cols / 2.0), img.rows));
    return img_piece_linear;
}
```



Рис. 8: Левая половина без изменений, правая растягивается в 2 раза по оси X

## 2.9 Проекционное отображение

Матрица Т:

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

Листинг 2.9 Проекционное отображение

```
cv::Mat projective_transform(cv::Mat img, double params[9]) {
    // params[9] = A, B, C, D, E, F, G, I
    cv::Mat T = (cv::Mat_<double>(3, 3) << params[0], params[1], params[2], params[3],
                 params[4], params[5], params[6], params[7], params[8]);
    cv::Mat img_projective;
    cv::warpPerspective(img, img_projective, T, cv::Size(img.cols, img.rows));
    return img_projective;
}
```

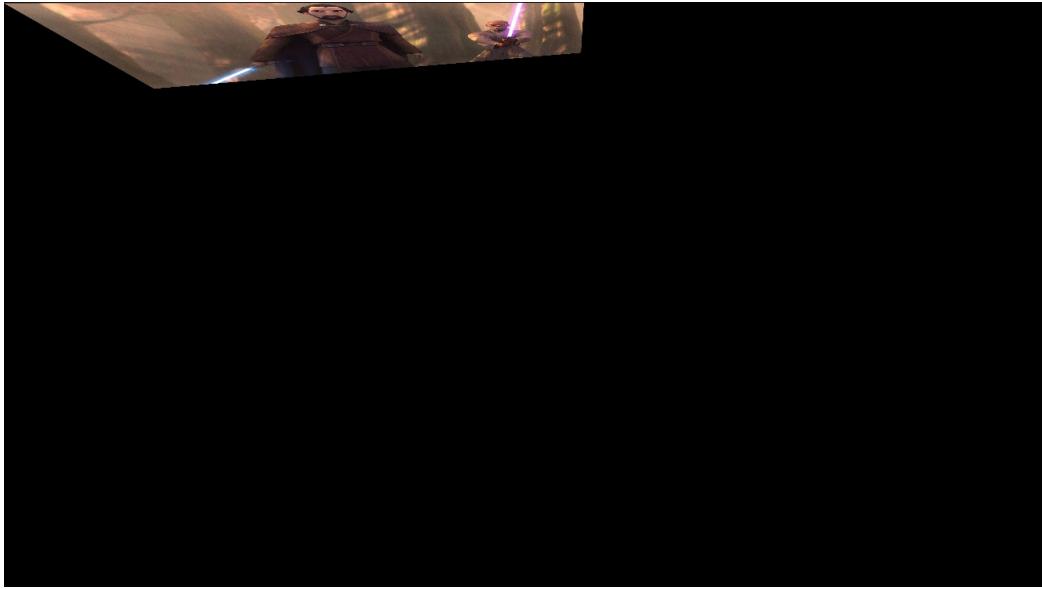


Рис. 9: Проекционное отображение

## 2.10 Полиномиальное отображение

Формула преобразования координат:

$$\begin{cases} x' = a_1 + a_2x + a_3y + a_4x^2 + a_5xy + a_6y^2 \\ y' = b_1 + b_2x + b_3y + b_4x^2 + b_5xy + b_6y^2 \end{cases} \quad (1)$$

Листинг 2.10 Полиномиальное преобразование

```
cv::Mat polinomial_transformation(cv::Mat img, double T[2][6]) {
    cv::Mat img_polinomial;
```

```

if (img.depth() == CV_8U) {
    img.convertTo(img_polynomial, CV_32F, 1.0 / 255);
}
else {
    img_polynomial = img;
}
std::vector<cv::Mat> img_BGR;
cv::split(img_polynomial, img_BGR);

for (int channel = 0; channel < img_BGR.size(); channel++) {
    img_polynomial = cv::Mat::zeros(img_BGR[channel].rows, img_BGR[channel].cols,
        img_BGR[channel].type());
    for (int x = 0; x < img_BGR[channel].cols; x++) {
        for (int y = 0; y < img_BGR[channel].rows; y++) {
            int xnew = int(round(T[0][0] + x * T[0][1] + y * T[0][2] + x * x * T[0][3] + x * y
                * T[0][4] + y * y * T[0][5]));
            int ynew = int(round(T[1][0] + x * T[1][1] + y * T[1][2] + x * x * T[1][3] + x * y
                * T[1][4] + y * y * T[1][5]));
            if ((xnew >= 0) && (xnew < img_BGR[channel].cols) && (ynew >= 0) && (ynew <
                img_BGR[channel].rows)) {
                img_polynomial.at<float>(ynew, xnew) = img_BGR[channel].at<float>(y, x);
            }
        }
    }
    img_BGR[channel] = img_polynomial;
}

cv::merge(img_BGR, img_polynomial);
if (img.depth() == CV_8U) {
    img_polynomial.convertTo(img_polynomial, CV_8U, 255);
}

return img_polynomial;
}

```

---

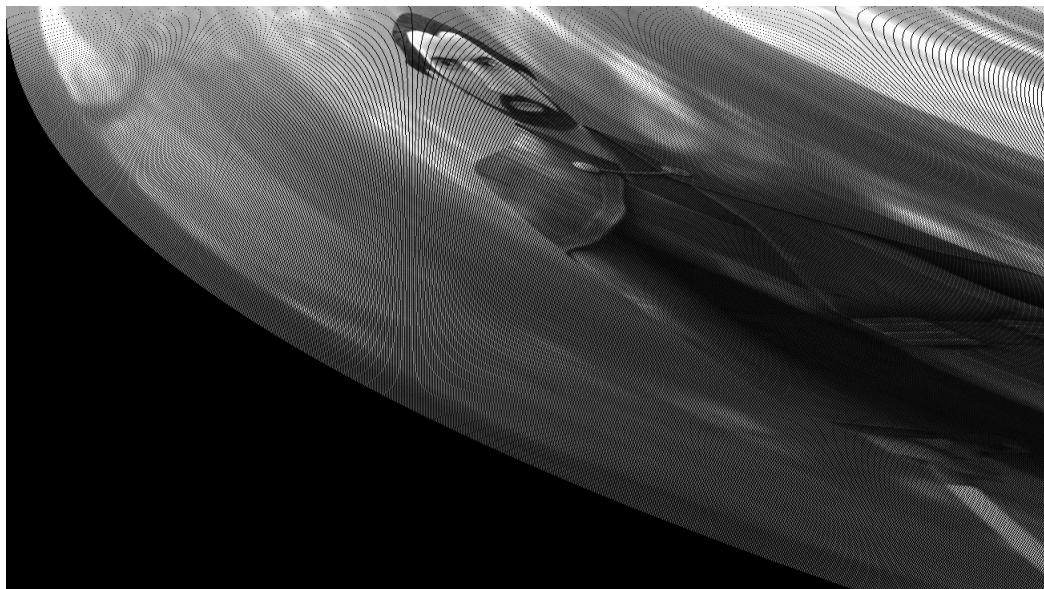


Рис. 10: Полиномиальное отображение

## 2.11 Синусоидальное искажение

### Листинг 2.11 Синусоидальное искажение

---

```
cv::Mat sin_transformation(cv::Mat img, double s) {
    cv::Mat u = cv::Mat::zeros(img.size(), CV_32F);
    cv::Mat v = cv::Mat::zeros(img.size(), 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 + s * std::sin(2 * M_PI * y / 90.0));
            v.at<float>(y, x) = float(y);
        }
    }
    cv::Mat sinusoid_image;
    cv::remap(img, sinusoid_image, u, v, cv::INTER_LINEAR);
    return sinusoid_image;
}
```

---



Рис. 11: Синусоидальное искажение  $s = 9$

### 3 Коррекция дисторсии

Дисторсия — это оптическое искажение, выражющееся в искривлении прямых линий. Для корректного анализа изображения лучше снизить влияние дисторсии на геометрию изображения. Если у нас есть прямой доступ к камере, которую мы хотим откалибровать, то с помощью маркеров ArUco можно провести коррекцию дисторсии. Если же доступа к камере у нас нет, можно попытаться исправить подушкообразную дисторсию наложением бочкообразной, а бочкообразную дисторсию - наложением подушкообразной. Приступим.

,c,

,c,,c,

### Листинг 3.1 Наложение бочкообразной дисторсии

---

```
cv::Mat barrel_distortion(cv::Mat img) {
    cv::Mat xi, yi;
    std::vector<float> t_x, t_y;
    for (int i = 0; i < img.cols; i++) {
        t_x.push_back(float(i));
    }
    for (int i = 0; i < img.rows; i++) {
        t_y.push_back(float(i));
    }

    cv::repeat(cv::Mat(t_x).reshape(1, 1), img.rows, 1, xi);
    cv::repeat(cv::Mat(t_y).reshape(1, 1).t(), 1, img.cols, yi);

    double xmid = xi.cols / 2.0;
    double ymid = xi.rows / 2.0;
    xi -= xmid;
    xi /= xmid;
    yi -= ymid;
    yi /= ymid;

    cv::Mat r, theta;
    cv::cartToPolar(xi, yi, r, theta);
    double F3(0.07), F5(0.12);
    cv::Mat r3, r5;

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

    cv::Mat u, v;
    cv::polarToCart(r, theta, u, v);
    u *= xmid;
    u += xmid;
    v *= ymid;
    v += ymid;

    cv::Mat img_barrel;
    cv::remap(img, img_barrel, u, v, cv::INTER_LINEAR);
    return img_barrel;
}
```

---



Рис. 13: Изображение после применения бочкообразной дисторсии

Как можно заметить, мы хоть и получили области с неопределенной интенсивностью по краям изображения, геометрических искажений стало меньше и дом выглядит действительно прямоугольным, что и было целью преобразования. С получившейся черной «рамкой» можно бороться с помощью геом.преобразований.

Перейдем к коррекции бочкообразной дисторсии.

### Листинг 3.2 Наложение подушкообразной дисторсии

```
cv::Mat pincushion_distortion(cv::Mat img) {
    cv::Mat xi, yi;
    std::vector<float> t_x, t_y;
    for (int i = 0; i < img.cols; i++) {
        t_x.push_back(float(i));
    }
    for (int i = 0; i < img.rows; i++) {
        t_y.push_back(float(i));
    }

    cv::repeat(cv::Mat(t_x).reshape(1, 1), img.rows, 1, xi);
    cv::repeat(cv::Mat(t_y).reshape(1, 1).t(), 1, img.cols, yi);

    double xmid = xi.cols / 2.0;
```

```

double ymid = xi.rows / 2.0;
xi -= xmid;
xi /= xmid;
yi -= ymid;
yi /= ymid;

cv::Mat r, theta;
cv::cartToPolar(xi, yi, r, theta);
double F3(-0.2);
cv::Mat r2;

cv::pow(r, 2, r2);
r += r2 * F3;

cv::Mat u, v;
cv::polarToCart(r, theta, u, v);
u *= xmid;
u += xmid;
v *= ymid;
v += ymid;

cv::Mat img_pincushion;
cv::remap(img, img_pincushion, u, v, cv::INTER_LINEAR);
return img_pincushion;
}

```

---



Рис. 14: Изображение с выраженной бочкообразной дисторсией



Рис. 15: Изображение после наложения подушкообразной дисторсии

Заметим, что наложение подушкообразной дисторсии действительно помогло выровнять изображение. Но, с другой стороны, привело к небольшим потерям данных по краям изображения. Неизменно, но и неудивительно, конечно.

## 4 «Склейка» изображений

### 4.1 Ручная склейка изображений



Рис. 16: Исходное изображение



Рис. 17: Левая часть изображения



Рис. 18: Правая часть изображения

#### Листинг 4.1 Ручная склейка изображений

```
cv::Mat glue_images(cv::Mat left_part, cv::Mat right_part) {
    int templ_size = 20;
    cv::Mat templ = left_part(cv::Rect(left_part.cols - templ_size - 1, 0, templ_size,
        left_part.rows));

    cv::Mat res;
    cv::matchTemplate(right_part, 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 final_img = cv::Mat::zeros(left_part.rows, left_part.cols + right_part.cols -
        max_loc.x - templ_size, left_part.type());
    left_part.copyTo(final_img(cv::Rect(0, 0, left_part.cols, left_part.rows)));
    cv::Mat right_part_for_glue = right_part(cv::Rect(max_loc.x + templ_size, 0, right_part.cols -
        max_loc.x - templ_size, right_part.rows));
    right_part_for_glue.copyTo(final_img(cv::Rect(left_part.cols, 0, right_part.cols - max_loc.x -
        templ_size, right_part.rows)));
    return final_img;
}
```



Рис. 19: Склейнное изображение

По сути, метод работает за счет приведения одной части в систему координат другой части изображения. Скейлка получилась достаточно незаметной, в том числе за счет аккуратно угаданного параметра `templ_size`.

## 4.2 Автоматическая склейка в OpenCV

А вот автоматическая склейка лажает (либо я в что-то сделал не так, тоже вариант), но склеить конкретно эти два изоображения не вышло, возможно, потому что общих точек на них маловато.

**Листинг 4.2** Автоматическая склейка в opencv

```
cv::Mat auto_glue_images(cv::Mat left_part, cv::Mat right_part) {
    cv::Ptr<cv::Stitcher> stitcher = cv::Stitcher::create(cv::Stitcher::PANORAMA);
    std::vector<cv::Mat> imgs;
    imgs.push_back(left_part);
    imgs.push_back(right_part);
    cv::Mat img_stitched;
    cv::Stitcher::Status status = stitcher->stitch(imgs, img_stitched);
    return img_stitched;
}
```

## 5 Заключение

**Выводы:** я применил несколько отображений и геометрических преобразований для решения задач пространственной коррекции изообразений. Опять немножко позже дедлайна, но и третья лаба почти готова, так что шансы нагнать как никогда велики. «Долго запрягаем, да быстро едем», дааа....

**Ответы на вопросы после лабораторной работы:**

1. Можно отразить относительно оси O<sub>x</sub> - эквивалентно повороту на 180. Хотя фактически такая матрица отражения - лишь частный случай матрицы поворота;
2. В соответствии с формулой  $t_{min} = \frac{(n+1)(n+2)}{2} = 15$  точек;
3. Связано с тем что область, содержащая информацию о нашем изображении меняет местоположение относительно исходного «холста». Что неизбежно приводит к появлению полностью черных областей. Можно бороться с этим с помощью дополнительного масштабирования изображения.