

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет «Информатика и вычислительная техника»

Кафедра «Математика и информатика»

**ОТЧЕТ**

по лабораторной работе №2 «Геометрические преобразования изображений»

Выполнил студент

Группы МСК21

Ветров Артем

Ростов-на-Дону

2024 г.

**Цель работы**

Освоение основных видов отображений и использование геометрических преобразований для решения задач пространственной коррекции изображений. Научиться применять геометрические и яркостные преобразования изображений, такие как сдвиг, отражение, масштабирование, поворот, аффинные и проективные преобразования. Дополнительно изучить эффекты, такие как "баррельная" дисторсия и объединение изображений.

**Теоретическое обоснование**

**Геометрические преобразования** позволяют изменять положение, ориентацию и размер изображения. К основным методам относятся:

1. **Сдвиг**: смещение изображения на заданное расстояние.
2. **Отражение**: создание зеркального отображения изображения.
3. **Масштабирование**: изменение размера изображения с сохранением пропорций.
4. **Поворот**: вращение изображения вокруг заданной точки.
5. **Аффинное преобразование**: линейное преобразование, которое сохраняет параллельность линий.
6. **Проективное преобразование**: позволяет изменять перспективу изображения.

**Обработка изображений** включает в себя трансформации, улучшающие качество и изменяющие геометрию изображения. Это может включать:

* **Яркостные трансформации**: построение и выравнивание гистограмм, которые помогают улучшить контраст и выявить скрытые детали.
* **Анализ**: профили и проекции для изучения яркостного распределения.
* **Специфические эффекты**, например, "баррельная" дисторсия, применяются для искусственного создания оптических искажений.

**Ход выполнения работы**

**Исходные изображения**





**Листинги программных реализаций**

**Сдвиг изображения**

Смещение изображения на 50 пикселей вправо и 100 вниз:

Mat shifted;

Mat shiftMat = (Mat\_<double>(2, 3) << 1, 0, 50, 0, 1, 100);

warpAffine(img, shifted, shiftMat, img.size());

showImage("Shifted Image", shifted);

**Отражение по оси X**

Создание зеркального отображения:

Mat reflected;

Mat reflectMat = (Mat\_<double>(2, 3) << 1, 0, 0, 0, -1, img.rows - 1);

warpAffine(img, reflected, reflectMat, img.size());

showImage("Reflected Image", reflected);

**Масштабирование изображения**

Изменение размера с коэффициентом масштабирования 1.5:

Mat scaled;

double scaleFactor = 1.5;

Mat scaleMat = (Mat\_<double>(2, 3) << scaleFactor, 0, 0, 0, scaleFactor, 0);

warpAffine(img, scaled, scaleMat, Size(int(img.cols \* scaleFactor), int(img.rows \* scaleFactor)));

showImage("Scaled Image", scaled);

**Поворот изображения**

Вращение изображения на 30 градусов:

Mat rotated;

double angle = 30.0;

Point2f center(img.cols / 2.0, img.rows / 2.0);

Mat rotateMat = getRotationMatrix2D(center, angle, 1);

warpAffine(img, rotated, rotateMat, img.size());

showImage("Rotated Image", rotated);

showImage("Scaled Image", scaled);

**Аффинное преобразование:**

Mat affineTransformed;

Point2f srcTri[3] = { Point2f(0, 0), Point2f(img.cols - 1, 0), Point2f(0, img.rows - 1) };

Point2f dstTri[3] = { Point2f(0, img.rows\*0.33), Point2f(img.cols\*0.85, img.rows\*0.25), Point2f(img.cols\*0.15, img.rows\*0.7) };

Mat affineMat = getAffineTransform(srcTri, dstTri);

warpAffine(img, affineTransformed, affineMat, img.size());

showImage("Affine Transformed Image", affineTransformed);

**Проективное преобразование:**

Mat projectiveTransformed;

Point2f srcQuad[4] = { Point2f(0, 0), Point2f(img.cols - 1, 0), Point2f(img.cols - 1, img.rows - 1), Point2f(0, img.rows - 1) };

Point2f dstQuad[4] = { Point2f(img.cols\*0.05, img.rows\*0.33), Point2f(img.cols\*0.9, img.rows\*0.2), Point2f(img.cols\*0.8, img.rows\*0.9), Point2f(img.cols\*0.2, img.rows\*0.7) };

Mat projectiveMat = getPerspectiveTransform(srcQuad, dstQuad);

warpPerspective(img, projectiveTransformed, projectiveMat, img.size());

showImage("Projective Transformed Image", projectiveTransformed);

**Баррельная дисторсия:**

void applyBarrelEffect(**const** Mat &I) {

int rows = I.rows;

int cols = I.cols;

*// Создание сетки*

Mat x\_coords = Mat::zeros(rows, cols, CV\_32F);

Mat y\_coords = Mat::zeros(rows, cols, CV\_32F);

**for** (int y = 0; y < rows; ++y) {

**for** (int x = 0; x < cols; ++x) {

x\_coords.at<float>(y, x) = x;

y\_coords.at<float>(y, x) = y;

}

}

*// Нормализация сетки*

float xmid = cols / 2.0;

float ymid = rows / 2.0;

x\_coords -= xmid;

y\_coords -= ymid;

*// Преобразование в полярные координаты*

Mat r, theta;

cartToPolar(x\_coords / xmid, y\_coords / ymid, r, theta);

*// Деформация (баррель-эффект)*

float F3 = 0.1;

float F5 = 0.12;

*// Преобразование r в Mat для применения pow*

Mat r\_pow3 = r.mul(r).mul(r); *// r^3*

Mat r\_pow5 = r\_pow3.mul(r).mul(r); *// r^5*

r = r + F3 \* r\_pow3 + F5 \* r\_pow5;

*// Обратное преобразование*

Mat u, v;

polarToCart(r, theta, u, v);

*// Восстановление масштаба*

u = u \* xmid + xmid;

v = v \* ymid + ymid;

*// Ремаппинг изображения*

Mat I\_barrel;

remap(I, I\_barrel, u, v, INTER\_LINEAR);

showImage("Barrel Effect", I\_barrel);

}

**Объединение изображений:**

void combineImages(**const** Mat &topPart, **const** Mat &botPart) {

int templ\_size = 10;

*// Вырезка шаблона*

Mat templ = topPart(Range(topPart.rows - templ\_size, topPart.rows), Range::all());

*// Поиск шаблона*

Mat res;

matchTemplate(botPart, templ, res, TM\_CCOEFF);

*// Поиск местоположения максимального значения совпадения*

double min\_val, max\_val;

Point min\_loc, max\_loc;

minMaxLoc(res, &min\_val, &max\_val, &min\_loc, &max\_loc);

*// Изменение размера botPart*

Mat botPart\_resized;

resize(botPart, botPart\_resized, Size(topPart.cols, botPart.rows));

*// Создание итогового изображения*

int result\_height = topPart.rows + botPart\_resized.rows - max\_loc.y - templ\_size;

Mat result\_img(result\_height, topPart.cols, topPart.type(), Scalar(0, 0, 0));

*// Копирование верхней части*

topPart.copyTo(result\_img(Range(0, topPart.rows), Range::all()));

*// Копирование нижней части*

Mat bot\_crop = botPart\_resized(Range(max\_loc.y + templ\_size, botPart\_resized.rows), Range::all());

bot\_crop.copyTo(result\_img(Range(topPart.rows, result\_img.rows), Range::all()));

showImage("Combined Image", result\_img);

}

**Комментарии**

* **Сдвиг изображения**: Использование матрицы трансформации позволяет легко перемещать изображение на заданное расстояние по горизонтали и вертикали, что полезно для подготовки данных или аугментации изображений.
* **Отражение изображения**: Создание зеркального отображения помогает анализировать симметрию объектов, а также генерировать дополнительные данные для обучения моделей машинного обучения.
* **Масштабирование изображения**: Изменение размера с коэффициентом масштабирования позволяет увеличить или уменьшить изображение без потери пропорций, что полезно при работе с данными разного разрешения.
* **Поворот изображения**: Вращение вокруг заданного центра применяется для выравнивания объектов или аугментации данных. Угол задается в градусах, что делает процесс интуитивно понятным.
* **Аффинное преобразование**: Линейное преобразование, сохраняющее параллельность линий, позволяет искажать изображение, сохраняя при этом основные свойства формы. Используется для моделирования различных углов обзора.
* **Проективное преобразование**: Позволяет изменять перспективу изображения, что имитирует эффект наклона камеры или плоскости. Применимо для подготовки изображений с разной геометрией.
* **Баррельная дисторсия**: Искусственное добавление эффекта искажения, напоминающего линзу. Это может быть полезно для моделирования реальных оптических искажений или для стилизации изображений.
* **Объединение изображений**: Функция объединяет две части изображения, используя шаблон для нахождения подходящего стыка. Это может применяться для обработки панорам или создания сложных композиций.

**Результирующие изображения**

**Сдвиг изображения**:



**Отражение изображения**:



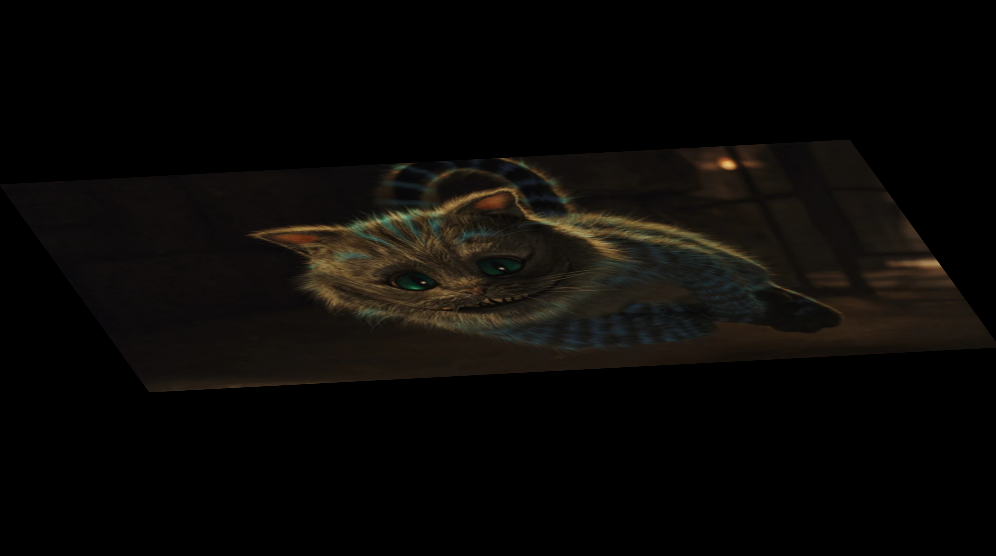
**Масштабирование изображения**:



**Поворот изображения**:



**Аффинное преобразование**:



**Проективное преобразование**:



**Баррельная дисторсия**:



**Объединение изображений**:



**Выводы о проделанной работе**

В ходе работы реализованы базовые методы обработки изображений в OpenCV, включая яркостные и геометрические преобразования. Освоены эффекты искусственных искажений (баррельная дисторсия) и работы с несколькими изображениями (объединение).

**Вопросы к защите лабораторной работе**

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

Поворот изображения можно выполнить, преобразовав координаты пикселей вручную. Для этого можно применить формулы поворота:

где *,* — исходные координаты пикселя, , — новые координаты после поворота, а — угол поворота. Значения для каждого пикселя вычисляются вручную, а результат записывается в новое изображение. Этот метод не требует использования встроенной матрицы поворота, но менее оптимален, так как требует циклов для обработки каждого пикселя.

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

Для геометрического преобразования порядка минимальное количество пар точек определяется как   
При :

Таким образом, требуется минимум **15 пар точек** для задания преобразования порядка 4.

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

**Причина**: После геометрического преобразования (например, масштабирования, поворота, сдвига) координаты пикселей не всегда соответствуют целым числам. Это приводит к "разрывам" в сетке пикселей, где значения остаются неопределенными.

**Решение**: Использование интерполяции для определения интенсивности в неопределенных пикселях. OpenCV предоставляет несколько методов интерполяции, таких как:

* **Билинейная интерполяция (INTER\_LINEAR)** — наиболее распространенный метод, который вычисляет значение как взвешенное среднее ближайших четырех пикселей.
* **Бикубическая интерполяция (INTER\_CUBIC)** — более точный метод, учитывающий ближайшие 16 пикселей.
* **Ближайший сосед (INTER\_NEAREST)** — простой метод, при котором пикселю присваивается значение ближайшего пикселя.  
  Выбор метода зависит от задач и требований к качеству изображения.