Отчет по лабораторной работе 5

Студента группы №3331506/00401 Орехова Алексея Михайловича

Код работы выложен на [гитхаб](https://github.com/alexeyorehoff/cv-lessons)

1. Задание:

1. Откалибровать собственную камеру при помощи доски aruco.

2. Откалибровать с помощью Calibration Toolbox (Matlab)

3. Сгенерировать маркер Aruco из понравившегося словаря

4. Написать программу, которая бы детектировала маркер на изображении и рисовать куб с основанием в виде маркера (куб должен быть спроецирован на плоскость изображения и иметь различные цвета ребер). Используйте захват видеопотока с камеры (VideoCapture), чтобы получать изображения.

1. Калибровка камеры методами Opencv

Для калибровки камеры было использовано два метода – с использованием встроенного инструмента OpenCV и с помощью camera calibration toolbox программы matlab.

Для калибровки встроенными функциями OpenCV, на этапе сборки библиотеки из исходников, были скомпилированы скрипты для калибровки камеры из модуля aruco. Для калибровки применены доска Cahuco, сгенерированная с помощью онлайн-генератора (<https://calib.io/pages/camera-calibration-pattern-generator>) и скрипт example\_aruco\_calibrate\_camera\_charuco.exe с параметрами -d=6 -h=8 -w=11 --ml=0.015 --sl=0.02 calibration.xml. При калибровке производилось по 10-15 снимков доски с разных ракурсов. В результате данные, полученные этим методом, вызвали сомнение в их правдоподобности. Например, фокусные расстояния по двум осям отличались 3 раза, составив 614 и 200 по осям x и y соответственно.

1. Калибровка камеры инструментом Matlab

Для этой калибровки использовалась шахматная доска, было сделано 13 снимков доски с разных ракурсов. В результате были получены более правдоподобные значения параметров оптики камеры:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Данные были записаны в .xml файл для дальнейшей обработки в разрабатываемой программе.

1. Отрисовка кубика

Детекция кубика производилась с помощью инструмента cv::aruco::ArucoDetector с выбранным словарем маркеров 5x5 соответствующим выбранному и напечатанному тестовому маркеру.

Положение маркеров в пространстве было определено с помощью функции cv::solvePnP, с учетом параметров камеры, полученных из xml файла с данными калибровки. Результатом работы алгоритма являются вектор положения и поворота маркера.

Отрисовка выполнялась в двух разных окнах двумя методами – встроенными OpenCV и с помощью библиотеки OpenGL. Для первого способа был создан массив вершин куба, а также вспомогательная функция cv::projectPoints, проецирующая вершины с ориентацией, заданной полученными ранее векторами, а также учитывающая параметры оптики камеры. По спроецированным точкам проводились отрезки, создававшие грани куба:

Изображение выглядит как человек, Человеческое лицо, одежда, Футболка

Автоматически созданное описание

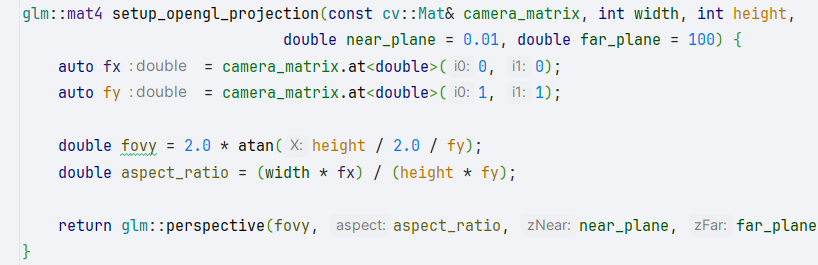
Отрисовка куба методами OpenGL же требовало дополнительных промежуточных шагов. Создание и обработка событий окна производилось с помощью библиотеки glfw. Был создан класс Window, куда была помещена инициализация glfw, создание окна и инициализация glew – библиотеки для работы с методами OpenGL, а также некоторые методы во время работы программы, такие как смена буфера экрана, проверка закрытия окна и очистка экрана.

Для того, чтобы отобразить на окне картинку с камеры создана простая шейдерная программа. На экран добавляются два треугольника, полностью закрывающие экран, после чего во фрагментном шейдере на них накладывается картинка с камеры как текстура. Текстура обновляется каждую итерацию цикла отрисовки с помощью метода draw\_bg() созданного класса glRenderer

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Перед отрисовкой куба требовалось настроить матрицу проекции OpenGL в соответствии с данными калибровки камеры. Для этого была создана функция setup\_opengl\_projection.



Для преобразования вектора поворота в матрицу была создана следующая функция:  
Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

С помощью функции cv::Rodrigues она преобразует вектор в матрицу поворота. Далее данные из матрицы типа cv::Mat\_<double, 3, 3> копируются матрицу библиотеки glm glm::dmat3. Данная матрица транспонируется, поскольку матрицы gl и cv отличаются порядком представления данных по столбцам и строкам, а для копирования применена функция memcpy, напрямую копирующая область памяти одной переменной в другую. Также, у библиотек opencv и opengl отличаются использующиеся системы координат, поэтому у получившейся матрицы требуется перевернуть направление оси Z. На этом этапе дополнительно применяется масштабирование.

Внутри вершинного шейдера положение точки преобразуется из локальной системы координат с СК экрана, после чего к ней применяется дисторсия. Алгоритм дисторсии основан на статье из документации OpenCV (<https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html>) и изменяет координаты точек в соответствии с их координатами на экране, а также коэффициентами дисторсии, полученными из основной программы.

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание Данный метод лишь изменяет положение конечных точек, и, в отличие от реальных искажений, не искривляет прямые. В пайплайне обработки графики такого возможно достичь на этапе растеризации, но насколько мне известно, в OpenGL нет механизма влиять на этот этап обработки. В результате был получен следующий результат:

Изображение выглядит как человек, Человеческое лицо, одежда, улыбка

Автоматически созданное описание Изображение выглядит как человек, текст, одежда, удержание

Автоматически созданное описание Изображение выглядит как снимок экрана, человек

Автоматически созданное описание

Можно заметить, что в сравнении с кубиком, спроецированным с помощью Opencv, левый кубик смещен относительно маркера, хоть и сохраняет правильное положение. Это может быть объяснено тем, что в разработанный алгоритм не удалось включить смещение оптического центра камеры. Кроме этого, встроенная функция проецирования OpenCV достаточно сложна и использует еще несколько механизмов дисторсий (<https://github.com/opencv/opencv/blob/ecbfc1bfd8d77e3dc5f3e0d981ce952e06c5f7e1/modules/calib3d/src/calibration.cpp#L522>) которые не представляется возможным применить в рамках лабораторной работы

Вывод:

Получены навыки применения калибровки камеры и работы с маркерами aruco в OpenCV. Создана программа, применяющая два вида отрисовки трехмерного объекта с использованием данных, полученных от детектора.