Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«Новгородский государственный университет

имени Ярослава Мудрого»

Институт электронных и информационных систем

Кафедра информационных технологий и систем

Допустить к защите

Заведующий кафедрой ИТиС

Профессор

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Петров Р.В.

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г.

**Программное обеспечение для калибровки параметров камеры**

Пояснительная записка к выпускной квалифицированной работе

по направлению 09.03.01

«Информатика и вычислительная техника»

НУОП.17001-01 81

Руководитель

\_\_\_\_\_\_\_\_\_\_ Назаров A.Г.

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2022 г.

Студент группы 8091

\_\_\_\_\_\_\_ Васильев И. В.

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2022 г.

Великий Новгород

2022

**АННОТАЦИЯ**

Пояснительная записка к диплому содержит:

* страниц – 86
* рисунков – 38
* таблиц – 2
* приложений – 5
* библиографических источников – 20

Разработанная программа полностью отвечает требованиям ТЗ.

**ANNOTATION**

The explanatory note provides:

* pages – 86
* figures – 38
* tables – 2
* attachments – 5
* bibliographic sources – 20

The developed program fully meets the requirements of the terms of reference.

Содержание

[ВВЕДЕНИЕ 4](#_Toc106117181)

[1. Анализ предметной области 5](#_Toc106117182)

[1.1. Определение камеры 5](#_Toc106117183)

[1.2. Математическая модель камеры 6](#_Toc106117184)

[1.3. Понятие дисторсии 10](#_Toc106117185)

[1.4. Модель стереокамеры 14](#_Toc106117186)

[1.5. Калибровка камеры 18](#_Toc106117187)

[1.6. Калибровка стереосистемы 22](#_Toc106117188)

[1.7. Калибровочные шаблоны 25](#_Toc106117189)

[1.8. Метрики качества калибровки 28](#_Toc106117190)

[1.9. Выявление аналогов 29](#_Toc106117191)

[2. Анализ требований 31](#_Toc106117192)

[2.1. Цель разработки 31](#_Toc106117193)

[2.1.1. Функциональные требования 31](#_Toc106117194)

[2.2. Проектирование 31](#_Toc106117195)

[2.2.1. Выбор языка программирование 31](#_Toc106117196)

[2.2.2. Выбор среды разработки 32](#_Toc106117197)

[2.2.3. Архитектура проекта 32](#_Toc106117198)

[2.3. Реализация 35](#_Toc106117199)

[2.3.1. Технические характеристики камеры 35](#_Toc106117200)

[2.3.2. Алгоритм загрузки изображений 36](#_Toc106117201)

[2.3.3. Алгоритм детекции шаблона 38](#_Toc106117202)

[2.3.4. Алгоритм калибровки 41](#_Toc106117203)

[2.3.5. Расчёт RMSE 47](#_Toc106117204)

[2.3.6. Инструменты анализа полученных результатов 48](#_Toc106117205)

[2.3.6. Пользовательский интерфейс 50](#_Toc106117206)

[ЗАКЛЮЧЕНИЕ 54](#_Toc106117207)

[СПИСОК ЛИТЕРАТУРЫ 55](#_Toc106117208)

[Приложение А – шаблон калибровки “шахматная доска” 57](#_Toc106117209)

[Приложение Б– шаблон калибровки “Круги” 58](#_Toc106117210)

[Приложение В– шаблон калибровки “Асимметричные круги” 59](#_Toc106117211)

[Приложение Г– шаблон калибровки “ChArUco” 60](#_Toc106117212)

[Приложение Д– программный код класса CalibrationProcessor 61](#_Toc106117213)

# ВВЕДЕНИЕ

Компьютерное зрение на данный момент считается достаточно молодой и быстроразвивающейся областью научных и прикладных исследований, основной целью которых является создание систем, способных извлекать из изображения информацию об объектах внешнего мира, полезную для дальнейшего использования. Обработка видеоинформации осуществляется на универсальных или специализированных компьютерах. Задачи обработки и анализа изображений открывают широкие перспективы автоматизации многих сфер человеческой деятельности. В качестве примеров можно отнести: контроль качества продукции на производстве, системы безопасности, биометрия, системы поиска изображений в базах данных, обнаружение, распознавание и сопровождение целей и др.

Специализированные видеокамеры являются основой для любых систем компьютерного зрения. От качества программно-аппаратного состава и настройки видеокамеры зависит точность получения информации из внешнего мира. Не существует возможности собрать такую видеокамеру, которая способна точно и без погрешности передать информацию об объекте реального мира. Для решения этой проблемы существует программная калибровка камеры, результатом работы которой, является математическая модель описывающая взаимосвязь оптики с электроникой.

Коммерческим заказчиком проекта выпускной квалификационнойработы является компания ООО«Квантово-оптические системы» (Далее заказчик). Заказчик разрабатывает системы компьютерного зрения, используя видеокамеры компании “Basler”, в данной работе в качестве типовой видеокамеры используется модель “daA1280-54um” (далее Камера). Для разрабатываемой продукции Заказчика необходимо выполнить калибровку камер, что способно повысить выходные характеристики разрабатываемых Заказчиком систем.

Программное обеспечение заказчика имеет ряд недостатков, которые способны ограничить Заказчика в возможности выявлять недостатки используемы методов калибровки камеры.

Целью данной выпускной квалификационной работы является создание программного обеспечения, способного усовершенствовать результаты калибровки и добавить возможность стерео калибровки существующего программного обеспечения калибровки Камер Заказчика.

# Анализ предметной области

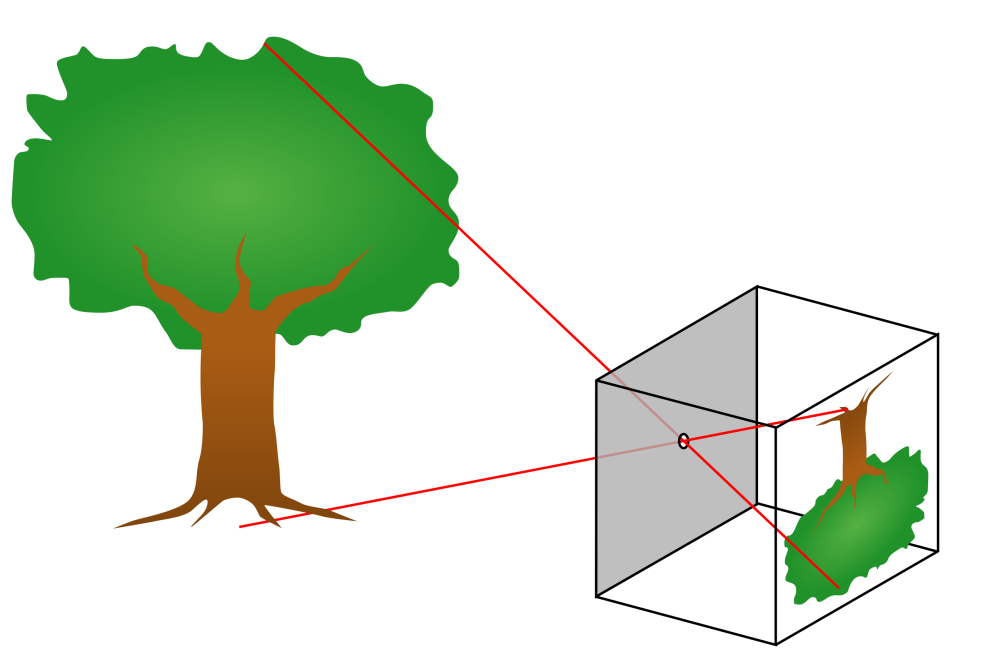
# Определение камеры

Камера – это устройство, позволяющее отобразить трёхмерное пространство в двухмерное изображение. Изображение формируется путём считывания камерой лучей света, падающих на светочувствительные ячейки матрицы камеры. Каждая ячейка при попадании на неё света вырабатывает электрический сигнал, пропорциональный интенсивности светового потока. При использовании информации о яркости света можно получить чёрно-белое изображение, но для этого необходимо накопить свет на светочувствительной матрице в течении определённого времени – это время называется временем экспозиции. Экспозиция влияет на яркость объектов на изображении, при недостаточном количестве света снимок получается слишком темным, а при слишком большом количестве света изображение получится слишком светлым.

Изображение представляет собой матрицу значений, каждая ячейка которой соответствует значению в градации серого цвета. Цветное изображение можно получить, если свет на светочувствительные ячейки матрицы камеры пропустить через фильтры красного, зелёного и синего цвета. В результате получается матрица значений с тремя каналами (RGB) для изображения. Одно значение пикселя соответствует трём значениям RGB.

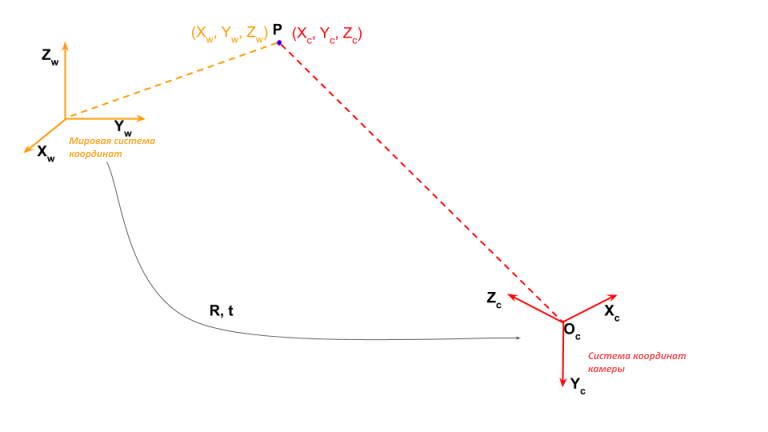
# Математическая модель камеры

Простой моделью того, как происходит подача света в камеру, является модель камеры-обскуры. Она предполагает, что свет исходит от сцены или удалённого объекта, но только один луч попадает в отверстие камеры, эта точка затем проецируется на поверхность изображения.



*Рисунок 1 Камера-обскура*

Что бы получить точку из трёхмерного пространства в координаты пикселя, необходимо разобрать геометрию формирования изображения.



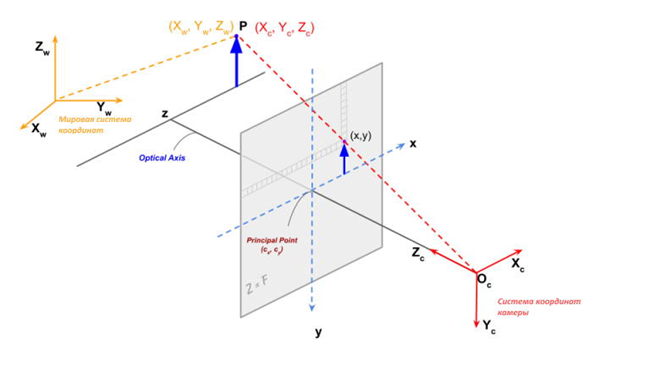
*Рисунок 2 Мировая система координат и система координат камеры, связанные поворотом и перемещением*

Мировая система координат – пространство, в котором находится камера. Начало координат - произвольная точка, направление осей которой соответствуют трёхмерным осям пространства. Точка P – произвольная точка в мировой системе координат, координаты которой задаётся по трём осям (). Что бы найти значение координаты, достаточно измерить расстояние по трём осям от точки P до начала координат.

В раннее упомянутое пространство поставим камеру в произвольное место (. Камера может смотреть куда угодно, следовательно она может вращаться относительно мировой системы координат. Вращение в 3D определяется тремя углами Эйлера . Углы Эйлера в авиации так же называются как тангаж, рысканье и крен.

Мировая система координат и координаты камеры связаны матрицей вращения R и вектором перемещения t - это способ описания математического оператора, который переводит произвольную координату из одной системы координат в другую. Это значит, что точка P, имеющая координаты () в мировой системе координат будет иметь другие значения (). Эти координаты связаны уравнением:

После того, как была получена точка в трехмерной системе координат камеры, применяя вращение и перевод к точкам мировых координат, появляется возможность спроецировать точку на плоскость изображения для вычисления местоположение точки на изображении.

*  
Рисунок 3 Проекция точки P на плоскость изображения*

Оптический центр (ось наблюдения) представлен с помощью . В действительности на плоскости формируется перевернутое изображение. Для математического удобства делаются все вычисления, как будто плоскость изображения находится перед оптическим центром, потому что изображение, считанное с матрицы, может быть тривиально повернуто на 180 градусов, чтобы компенсировать инверсию. На практике даже этого не требуется. Реальная светочувствительная матрица камеры читается, начиная с самого нижнего ряда в обратном порядке (справа налево), а затем снизу вверх для каждого ряда. Такой метод автоматически формируется правильное изображение и по вертикали и по горизонтали. Так что на практике нет необходимости переворачивать изображение.

Плоскость изображения размещается на расстоянии F (фокусное расстояние) от оптического центра.

Используя формулу подобия треугольников можно посчитать проектное изображение (x,y) 3D-точки ():

,

Приведённые выше уравнения перепишем в матричную форму:

Матрица K называется внутренней матрицей и содержит внутренние параметры камеры.

Приведенная выше матрица показывает только фокусное расстояние. Однако пиксели в светочувствительной матрице камеры могут быть не квадратными, что приводит к другому соотношению сторон вдоль оси x и y. Вот почему фокусное расстояние стоит разделить на два параметра . Оптический центр  камеры может не совпадать с центром системы координат изображения. Учитывая недостатки, матрица камеры принимает следующий вид:

где,

Проецирование трехмерной точки в мировой системе координат на пиксельные координаты камеры выполняется в два этапа.

1. Трехмерная точка преобразуется из мировых координат в координаты камеры с помощью внешней матрицы, которая состоит из вращения и перемещения между двумя системами координат.

2. Новая трехмерная точка в системе координат камеры проецируется на плоскость изображения с помощью встроенной матрицы, которая состоит из внутренних параметров камеры, таких как фокусное расстояние, оптический центр и т. д.

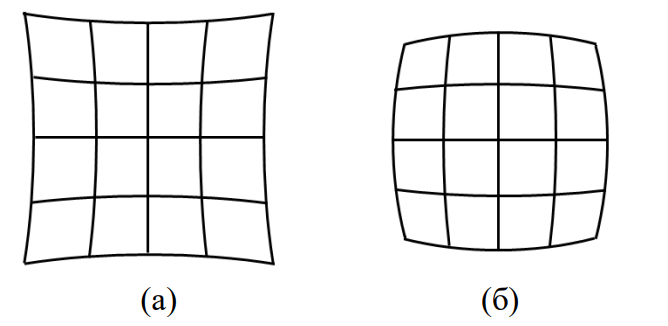
# Понятие дисторсии

Оптическая система камеры вносит нелинейные искажения, называемые дисторсией. Наличие дисторсии не только приводит к искривлению прямых линий, но и приводит к грубым ошибкам при измерениях по сделанному снимку.

Дисторсия делится на два типа:

1. Радиальная – связанная с кривизной линзы
2. Тангенциальная – связанная с наклоном оптической системы объектива относительно светочувствительного сенсора.

Радиальная дисторсия заключается в смещении точки на изображении в сторону к оптической оси или от нее относительно ее истинного положения. В случае если смещение направлено в сторону оптической оси, полученное искажение принято называть бочкообразным. В противном случае – подушкообразным.



*Рисунок 4 Радиальная дисторсия: (а) – Радиально подушкообразная, (б) – Радикальная бочкообразная*

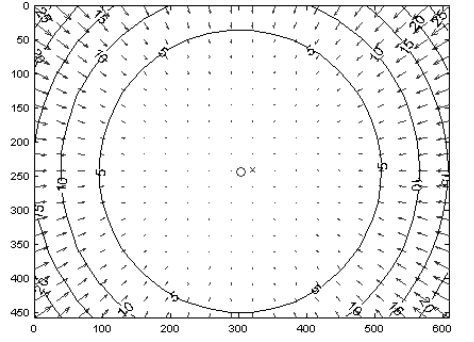
Вернёмся к обозначениям на рисунок 3. Координаты точки на изображении являются точкой идеального положения рассматриваемой модели камеры-обскуры. Наличие объектива не позволяет использовать вышеупомянутую математическую модель, что приводит к геометрическим искажениям (радиальном положением точки). Новая точка будет равна следующим координатам .

Предполагая, что центр дисторсии совпадает с началом координат, необходимо ввести радиальное расстояние от начала координат до расположения точки и радиальное расстояние от начала координат до реально расположенной точки .

Наиболее известная модель радиальной дисторсии имеет вид:

Где

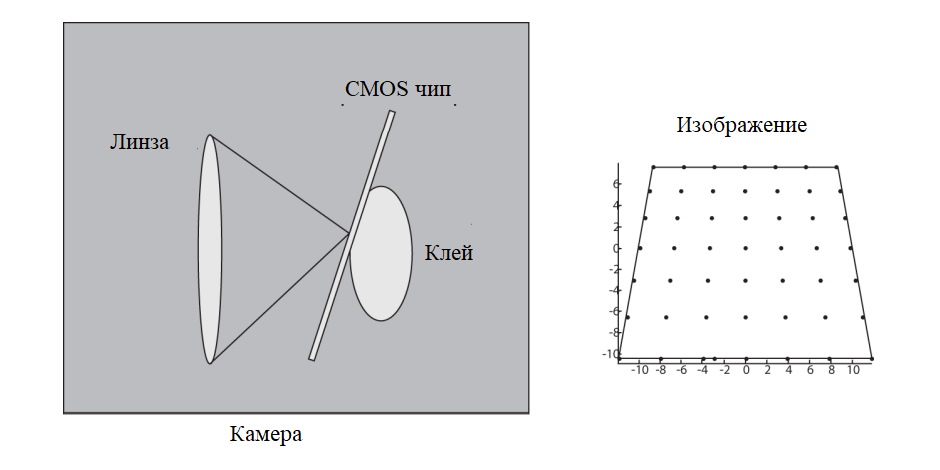
На рисунке 5 показан дисплей размещения прямоугольной сетки из-за радиальной деформации.



*Рисунок 5 График радиальных искажений для объектива камеры*

Стрелки показывают, где точки на внешней прямоугольной сетке смещаются в радиально искаженном изображении (рисунок 5).

Тангенциальная дисторсия, связанная с наклоном плоскости датчика, выражается в проекционных искажениях регистрируемого изображения.



*Рисунок 6 Возникновение тангенциальной дисторсии*

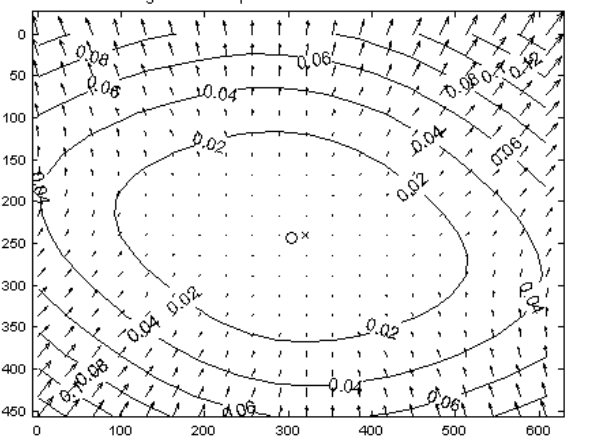
Тангенциальная дисторсия возникает, когда объектив не полностью параллелен плоскости изображения (рисунок 6).

Касательные тангенциального искажения характеризуются двумя параметрами: и

,

.

На рис. 6 показано влияние тангенциального искажения на лицевую поверхность. Точки смещаются эллиптически как функция местоположения и радиуса. Стрелки показывают, где точки на внешней прямоугольной сетке смещаются в тангенциально искаженном изображении (рисунок 7).



*Рисунок 7 График тангенциальных искажений для объектива камеры*

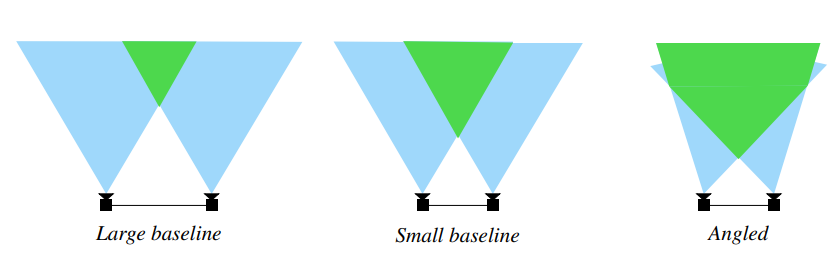
Таким образом, всего требуется пять коэффициентов дисторсии, они обычно объединены в один вектор искажения; это матрица 5 × 1, содержащая , , , , .

# Модель стереокамеры

Стерео зрение - это метод компьютерного зрения, используемый для восприятия глубины сцены путём получения изображений со стереокамеры. Стереокамера - две камеры, важным критерием для которых является общая фиксированная ось, на которой расположены эти камеры.

При наблюдении за объектом, стереокамера должна “видеть” объект в своём поле зрения. Поле зрения напрямую связано с расстоянием между двумя камерами, базовой линией и углом между камерами. Оценка глубины возможна только при перекрывающем поле зрения между двумя камерами.

На рисунке 8 показаны три различных установки, где перекрывающее поле зрения показано зелёным цветом. Перекрывающее поле зрения будет уменьшаться с увеличением базовой линии, кроме того, слепое пятно будет расширяться. Решением для уменьшения слепой зоны является расположение камер под углом друг к другу.

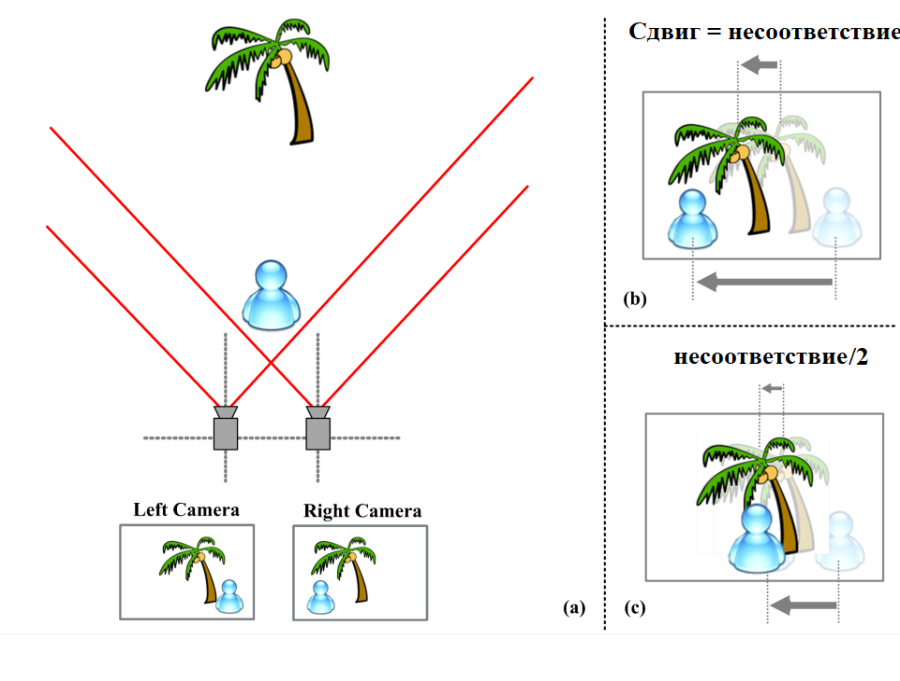


*Рисунок 8 Поле зрения при разных стереонастройках*

Базовая линия прямо пропорциональна горизонтальному расстоянию от крепления до объекта, таким образом, стереосистемы с меньшей базовой линией достигают лучших результатов при обнаружении объектов на средней дистанции, тогда как более широкая базовая линия предпочтительнее при больших расстояниях. Для оценки размера базовой линии можно использовать правило :

* Если вы знаете расстояние между камерой и объектом съёмки, вы можете рассчитать базовое значение по соотношению . Например, если расстояние составляет 72 дюйма, то основание должно быть 2.4 дюйма.
* Если вы знаете базу, вы можете рассчитать расстояние до объекта. Например, если объективы камер расположены на расстоянии 3 дюйма друг от друга, то основной объект съёмки должен находиться на расстоянии около 90 дюймов (30 x 3 дюйма).

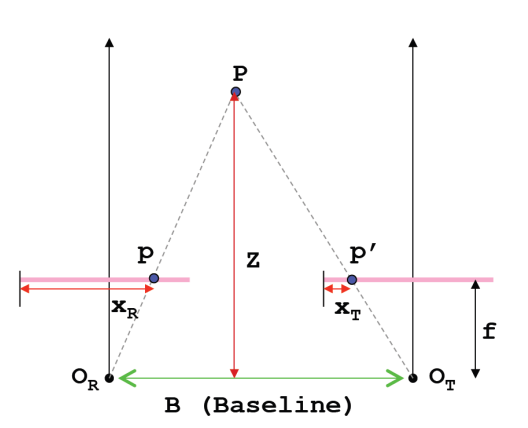
Для рассмотрения модели стереокамеры, необходимо ввести понятие несоответствия. Несоответствие (disparities) – это сдвиг, который можно наблюдать между точкой на изображении левой камеры и её аналог на изображении правой камеры. Чем ближе объект от камеры, тем больше будет несоответствие. Глубина объекта в сцене увеличивается по мере уменьшения несоответствия.



*Рисунок 9 Концепция несоответствия*

В рассматриваемой модели стереокамеры используется модель камера-обскура. Глубина трёхмерной точки P получается следующим образом:  
на рисунке 10 представлен вид сверху на две камеры с центрами и , находящиеся друг от друга на расстоянии B - называемым базовой линией. Индексы R и T расшифровываются как “Reference camera” и "Target camera". Обе камеры имеют одинаковое фокусное расстояние f, а розовые линии представляют плоскость изображения. Точки и соответствуют проекции P на R и T камерах. С помощью подобных треугольников и получим следующее соотношение:

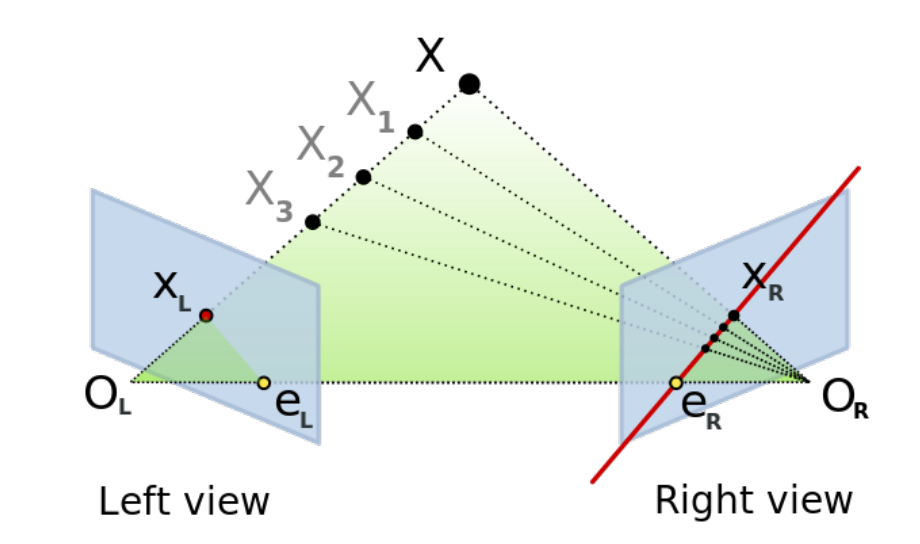
Где - несоответствие



*Рисунок 10 Проекция точки P с помощью двух идеально выровненных камер-обскура*

Однако в реальной жизни идеально выровнять камеры невозможно, поэтому необходимо ввести понятие эпиполя. Эпиполь – это точка пересечения линий, соединяющей два оптических центра базовой линией с плоскостью изображения. Точки рисунке 11 являются эпиполями левого и правого изображения.

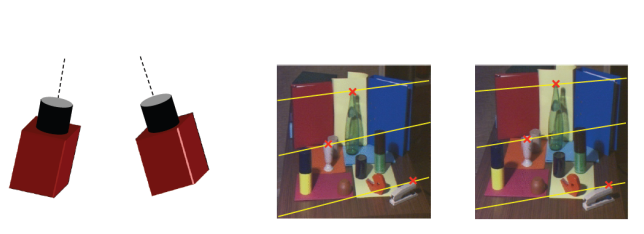
Эпиполярная плоскость - плоскость трёхмерной точки X и оптическими центрами. Плоскость представлена зелёным цветом.

**

*Рисунок 11 Эпиполи левой и правой камеры, оптические центры которых не совмещены*

Эпиполярная линия представляет собой прямую линию пересечения эпиполярной плоскости с плоскостью изображения. Эпиполярная линия в одной камере проходит через свой оптический центр и точку изображения в другой камере. Все эпиполярные линии пересекаются в эпиполе. На рисунке 10 эпиполярная линия обозначена красным цветом. Соответственно эпиполярная линия в левом изображении равна .

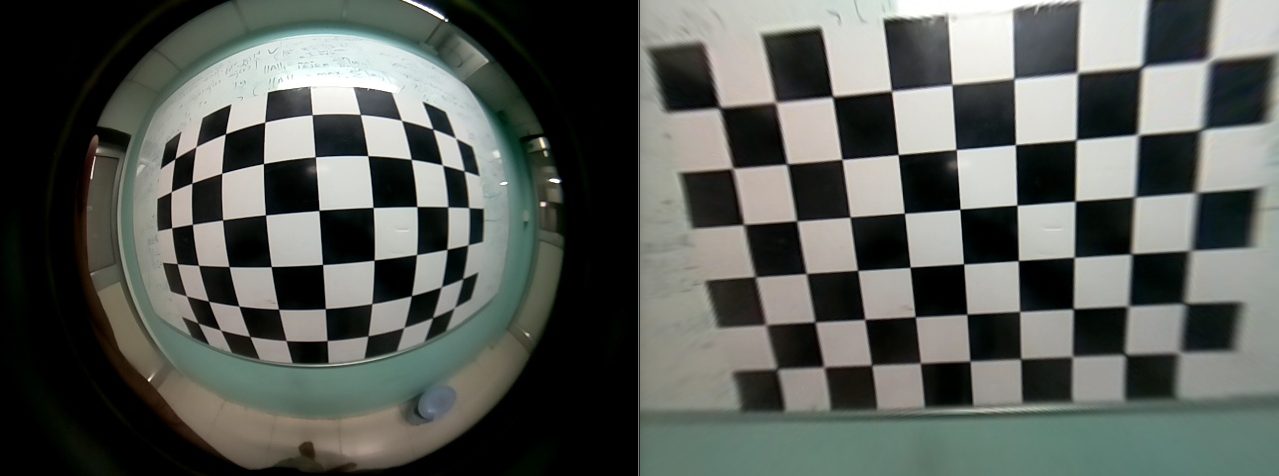
В такой конфигурации соответствующие пиксели не расположены на одной горизонтальной линии, но всё ещё расположены на эпиполярной линии как на рисунке 12.



*Рисунок 12 Эпиполярные линии в стереопаре, снятые на невыровненные камеры*

# Калибровка камеры

Калибровка камеры используется на начальном этапе решения многих задач компьютерного зрения. Задача калибровки состоит в получении внутренних и внешних параметров камеры, а так же коэффициентов дисторсии. Использование откалиброванной камеры даёт более точное соответствие соотношению между естественными единицами измерения камера (пикселями) и единицами измерения физического мира (метрами).



*Рисунок 13 Наглядное применение калибровки камеры*

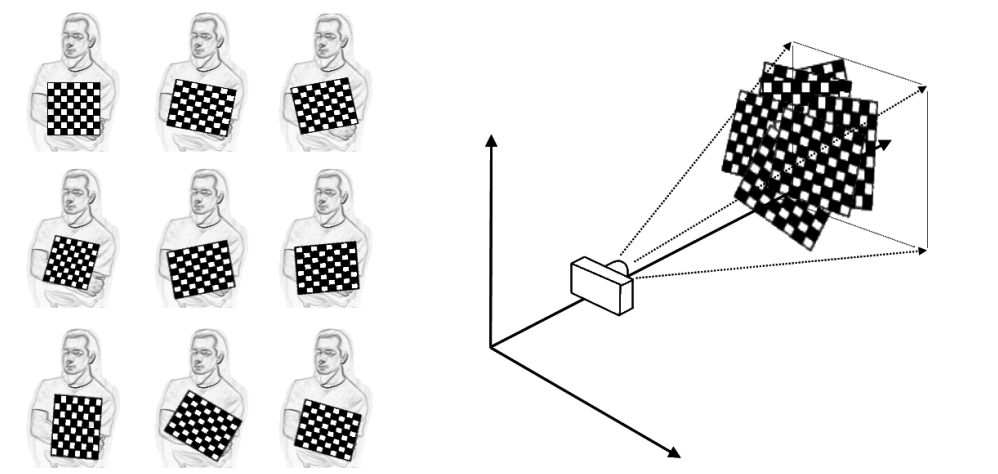
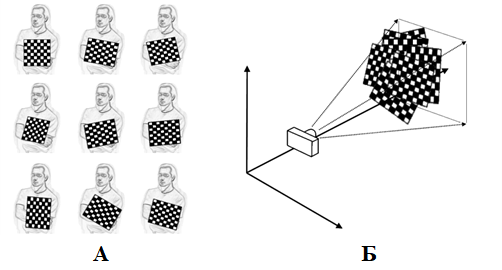
Слева изображение камеры с линзой "рыбьего глаза", справа исправленное левое изображение с помощью калибровки

К методам калибровки камеры относятся:

* По шаблону - даёт возможность полностью контролировать процесс формирования изображение. Лучший способ выполнить такую калибровку – сделать несколько изображений объекта или шаблона известных размеров с разных точек зрения. Такие шаблоны называются калибровочные шаблоны.
* На основе глубокого машинного обучения – позволяет в условиях ограниченной возможности управления настройками изображения (например, существует лишь единственное изображение на сцене) выполнить калибровку, использую методы глубокого машинного обучения.

Алгоритм калибровки камеры с фиксированным объективом:

1. Определение действительных мировых координат 3D точек шаблона известного размера.
2. Захват нескольких изображений калибровочного шаблона с разных точек наблюдения
3. Поиск всех действительных координат 3D точек в точки на изображении различного позиционирования
4. Выполнение калибровки с использованием ранее найденного массива точек
5. Анализ калибровки и выявление оптимальных изображений.

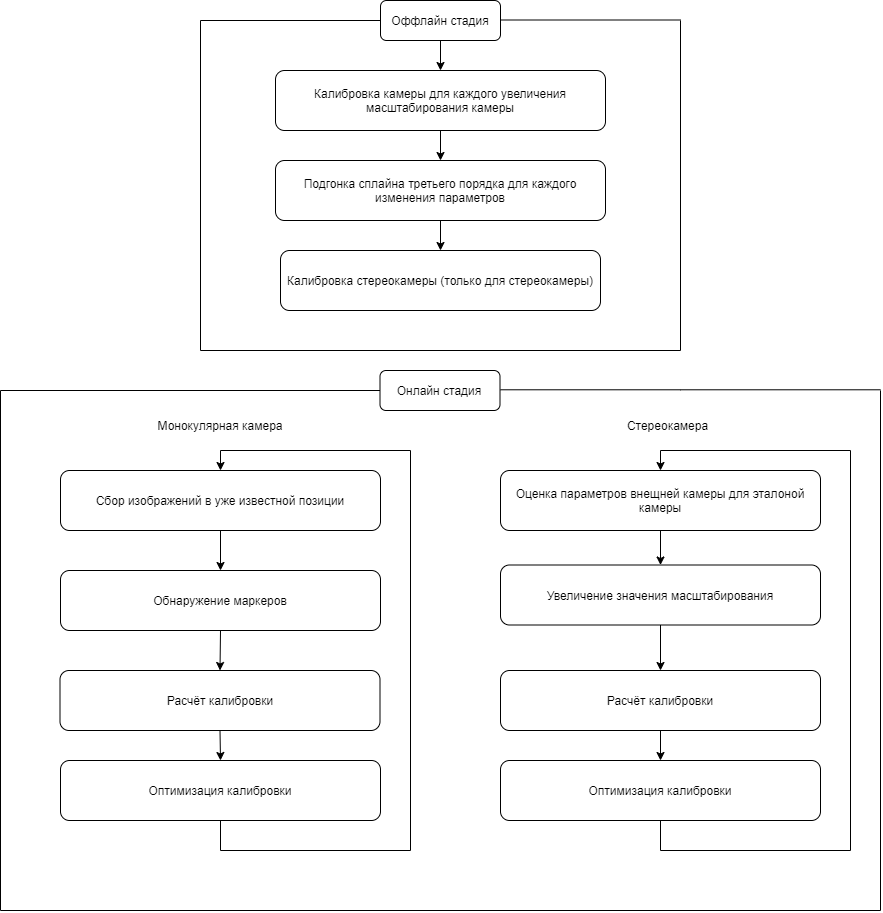


*Рисунок 14 Процесс съёмки*

Калибровочный шаблон держат в разных ориентациях (рисунок 14 - А), этого достаточно, что бы получить информацио о местонахождения этих изображений в глобальной системе координат относительно камеры (рисунок 14– Б)

Так же встречаются камеры с динамическим объективом, процесс калибровки таких камер разделяется на два процесса, автономная калибровка камеры и оценка параметров онлайн-камеры. В автономном процессе изменение внутренних параметров камеры моделируется путём калибровки внутренней камеры параметра для каждого значения масштабирования. В процессе оценки параметров онлайн-камеры внутренние и внешние параметры оцениваются с использованием предварительной калибровки.

Метод калибровки динамического объектива основан на монокулярной камере и стереокамере:



*Рисунок 15 Блок-схема оценки параметров камеры для масштабируемой камеры*

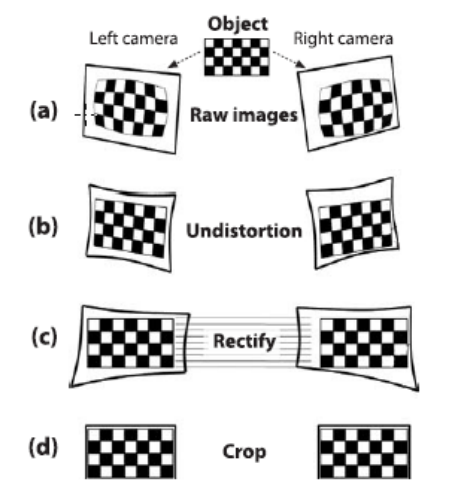
На точность калибровки камеры сильно влияют внешние факторы. Для оптимальной калибровки следуют придерживаться следующих рекомендаций:

* В помещении, где проводится съемка, должен быть фиксированный уровень освещённости
* Высокая точность изготовления калибровочного шаблона
* Калибровочный шаблон должен иметь размер, покрывающий половину общей площади на изображении с камеры
* Камера должна быть сфокусирована на калибровочном изображении
* В камере должны быть отключены функции авто-фокуса и авто-экспозиции
* Изображение необходимо собирать с разных областей наклона
* Точки калибровочного шаблона должны полностью покрывать область изображения, и стремиться к равномерному покрытию
* Изображения должны быть как фронтальные, так и сделанные с наклоном шаблона до +/- 45 градусов по горизонтали и по вертикали. Большой наклон не является хорошей идеей, поскольку начинает страдать локализация обнаружения точек шаблоне.
* Удалите плохие наблюдения. Внимательно осмотрите ошибки перепроецирования. Как за просмотр, так и за функцию. Если какие-либо из них отображаются как выбросы, исключите их и повторите калибровку.
* Получение низкой ошибки воспроизведения не означает хорошую калибровку камеры, а просто указывает на то, что предоставленные данные/доказательства могут быть описаны с помощью используемой модели. Это может быть связано с переоснащением. Неопределенности параметров являются показателями того, насколько хорошо выбранная модель камеры была ограничена.
* Проанализируйте отдельные ошибки перепроецирования. Их направление и величина не должны коррелировать с положением

# Калибровка стереосистемы

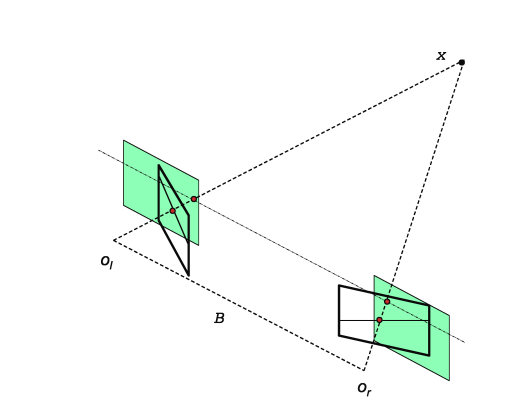
Калибровка стереосистемы отвечает за установление внутренних и внешних параметров каждой камеры и их параметры дисторсии, а так же преобразований между их системой отсчёта. Все эти параметры нужны для решения проблемы с согласованием стерео и возможности перепроектирования точки в трёхмерном мире на основе несоответствия.

Отличительной особенностью метода стерео калибровки от моно-калибровки - является наличие преобразований между системой отсчёта двух камер. Для получения параметров для каждой камеры, выполняется съёмка шаблона, после чего выполняется сначала моно-калибровка для каждой камеры.

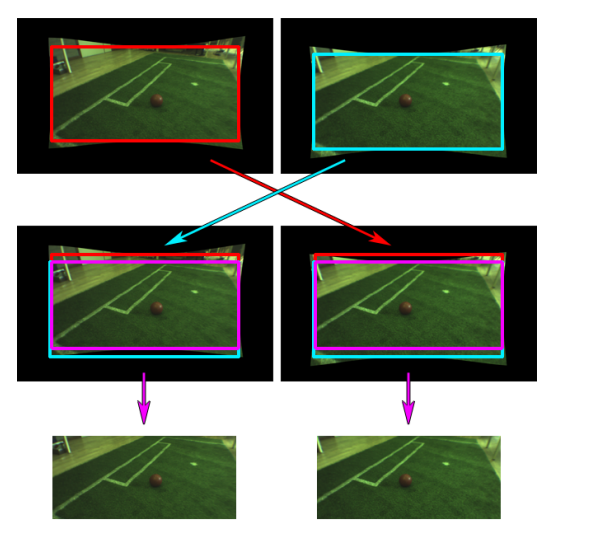


*Рисунок 16 Основные этапы исправления стереопары изображений*

После стерео калибровки необходимо выполнить исправление (rectification) стереопары изображений с целью виртуально совместить камеры в стереосистему и убрать искажения, что бы соответствующие пиксели располагались на одной и той же строке на двух изображениях. Исправление стереопары – это процесс переназначения точек с плоскостей изображения несовмещенных камер на новые плоскости изображения, повёрнутые таким образом, что эпиполярные линии новых повёрнутых плоскостей должны быть горизонтальными, а соответствующие линии должны располагаться в одной строке на обоих изображениях, как показано рисунке 17, где плоскости зелёного изображения – плоскости, которые были исправлены.

  
*Рисунок 17 Невыровненные и выровненные плоскости изображения*

После получения выровненных изображений, они обрезаются, для содержания только известных пикселей. Прямоугольная область интереса определяется для каждого изображения, которое охватывает как можно большую часть изображения, без включения неизвестных значений пикселя при пересечении между интересующей областью обоих изображений, как показано на рисунке 18.



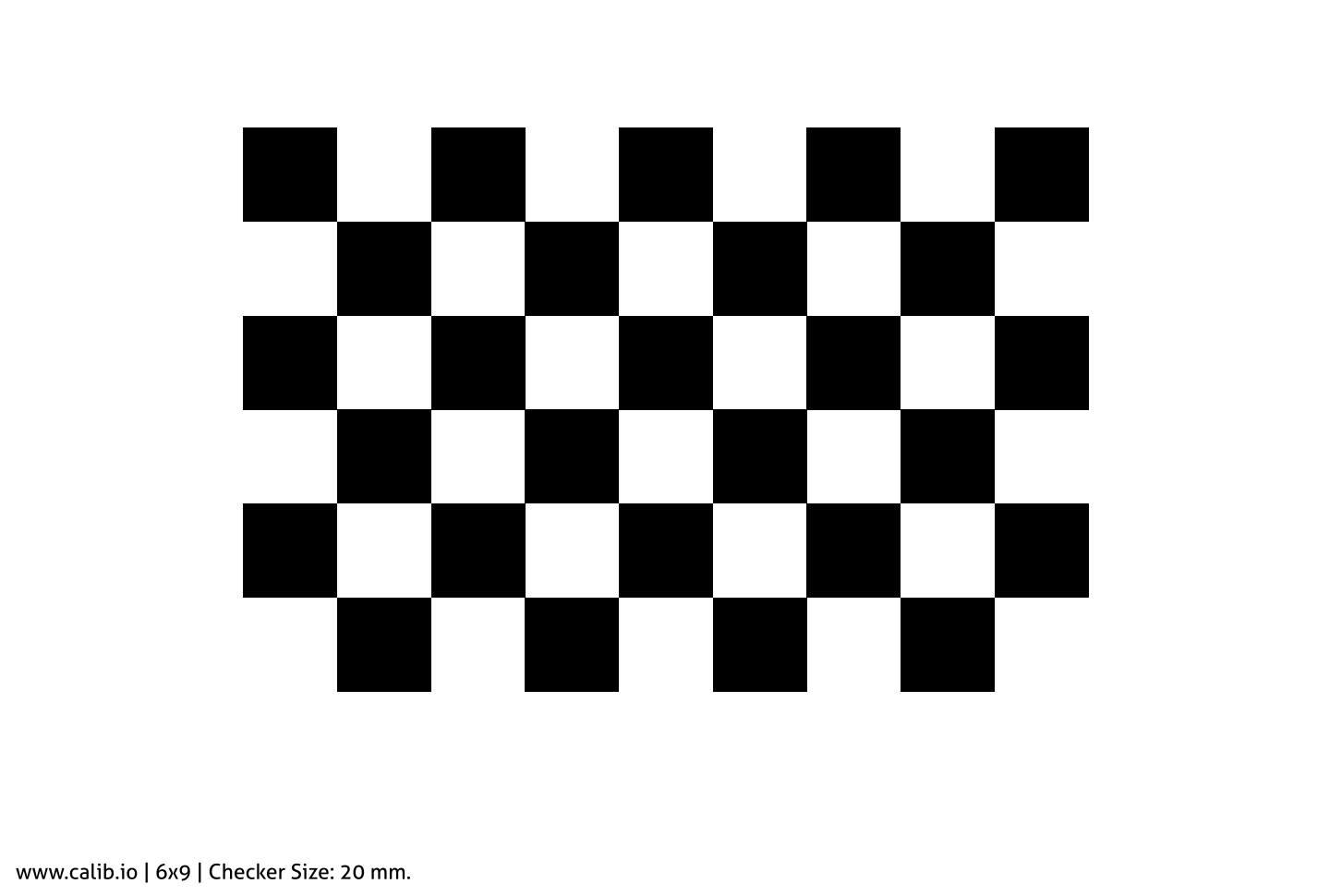
*Рисунок 18 Обрезка выпрямленной пары стереоизображений*

# Калибровочные шаблоны

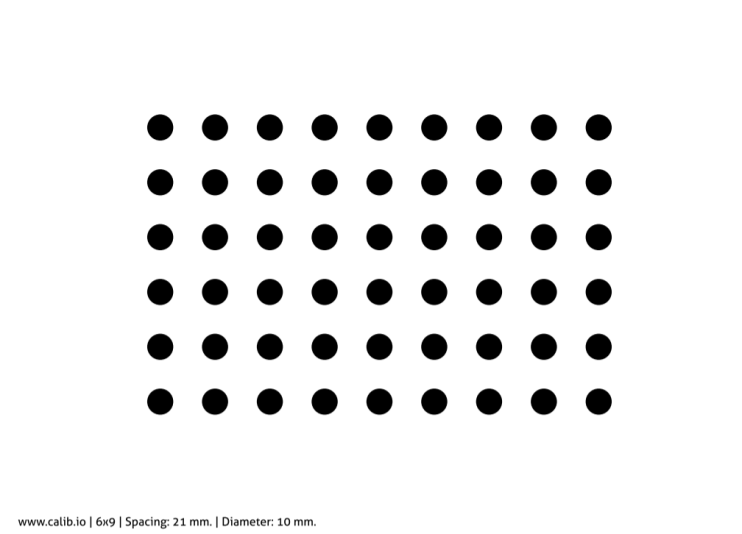
Калибровочные шаблоны различаются по типу узору и индивидуальных алгоритмов их обнаружения, бывают следующих видов:

* Шахматная доска
* Симметричные круги
* Асимметричные круги
* ChArUco

Шаблон шахматной доски является распространённым в использовании калибровочным шаблоном. Контрольные точки для этого шаблона являются углами, которые лежат в шахматной доске. Поскольку углы чрезвычайно малы, они являются часто инвариантными к перспективе и искажению объектива. Необходимо использовать шахматную доску, которая содержит четное число квадратов вдоль одного ребра и нечетного числа квадратов вдоль другого ребра с двумя черными угловыми квадратами вдоль одной стороны и двумя белыми угловыми квадратами на противоположной стороне. Это позволяет приложению определить ориентацию шаблона и источника.

****

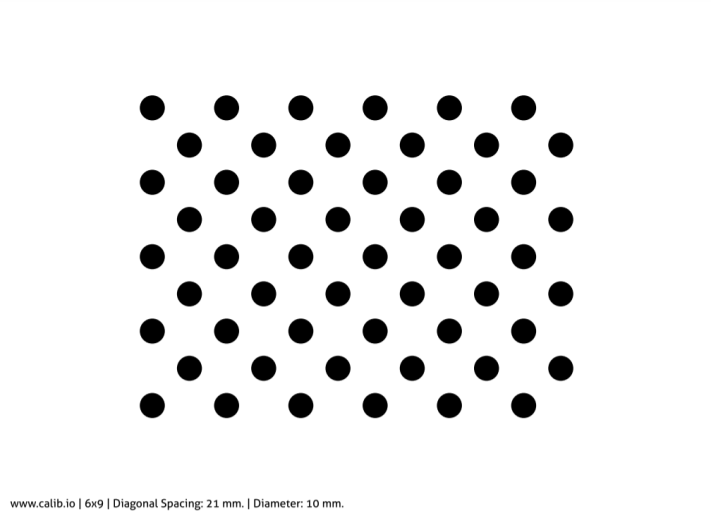
*Рисунок 19 Пример калибровочного шаблона шахматной доски*

Альтернативой шахматной доске является круговая сетка. Концептуально круговая сетка похожа на шахматную доску, за исключением того, что вместо массива чередующихся черных и белых квадратов, доска содержит массив черных кружков на белом фоне. Круговые шаблоны сетки, иногда называемые grid of circles, являются классом калибровочных шаблонов, которые используют равномерно распределенные круги, чтобы сформировать структуру сетки. Они широко классифицируются в два типа: ****симметричные и асимметричные шаблоны.

*Рисунок 20 Пример калибровочного шаблона “симметричные круги”*

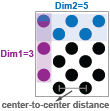
Особенности шаблона симметричные круги:

* Круги располагаются равномерно в строках и столбцах
* Размерности измеряются в количестве кругов как [высота, длинна], где высотой является количеством кругов в одной строке, а длинной является количеством кругов в одном столбце.
* Калибровочный шаблон на данного типа не может использоваться для калибровки стереокамеры из-за неоднозначности на 180 градусов.

****

*Рисунок 21 Пример калибровочного шаблона “асимметричные круги”*

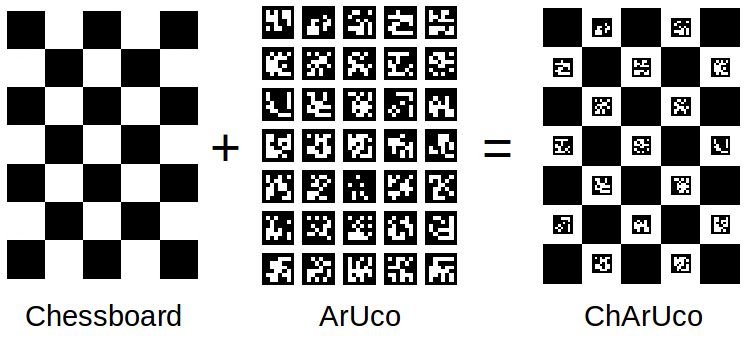
Особенности шаблона ассиметричные круги:

* Каждая вторая строка кругов возмещена наполовину расстояние столбца между соседними элементами строки.
* Размерности измеряются в количестве кругов как [dim1 dim2], где dim1 является количеством кругов по измерению, которое содержит то же количество кругов в каждой строке или столбце, и dim2 является количеством кругов через два смежных столбца (или строки) в размерности, где эти два столбца (или строки) содержат неравное количество кругов.

*Рисунок 22 Пример размерности шаблона*

* Большая плотность точек для того же кругового радиуса.
* Может использоваться для калибровки стереокамеры. Отсутствует неоднозначность на 180 градусов.

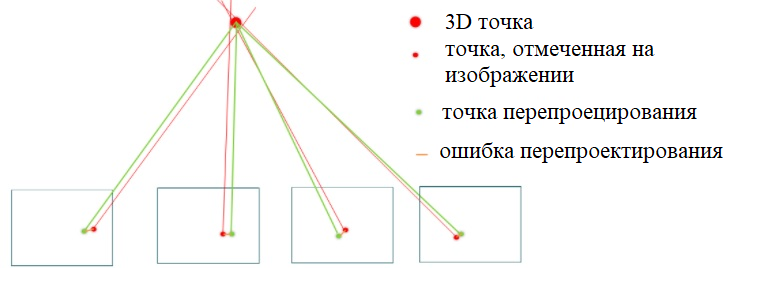
У шахматной доски есть проблема, которая не может быть решена из-за структуры шаблона - она должна быть полностью видимой, то есть все угла шахматной доски должны лежать в поле видимости изображения. Платы ArUco отличаются быстрым обнаружением и универсальностью, однако одна проблема с маркировкой ArUco состоит в том, что даже после уточнения субпикселей точность их угловых положений не очень высока. Для решения этой проблемы было представлено объединение шахматной доски и ArUco маркеров – ChArUco.



*Рисунок 23 Пример калибровочного шаблона “ChArUco”*

# Метрики качества калибровки

Точность калибровки камеры оценивается по среднеквадратичной ошибки обратной проекции по всем точкам детекции, изображений которых участвуют в процессе калибровки. Для нахождения ошибки, берётся положение точки, как минимум на двух изображениях камеры,  вычисляются ее трехмерные координаты с использованием внутренних и внешних параметров камеры. Как только трехмерные координаты точки вычислены, трехмерная точка повторно проецируется на изображения. Расстояние между точкой детекции и точкой обратной проекции на одном изображении является ошибкой обратного проецирования.



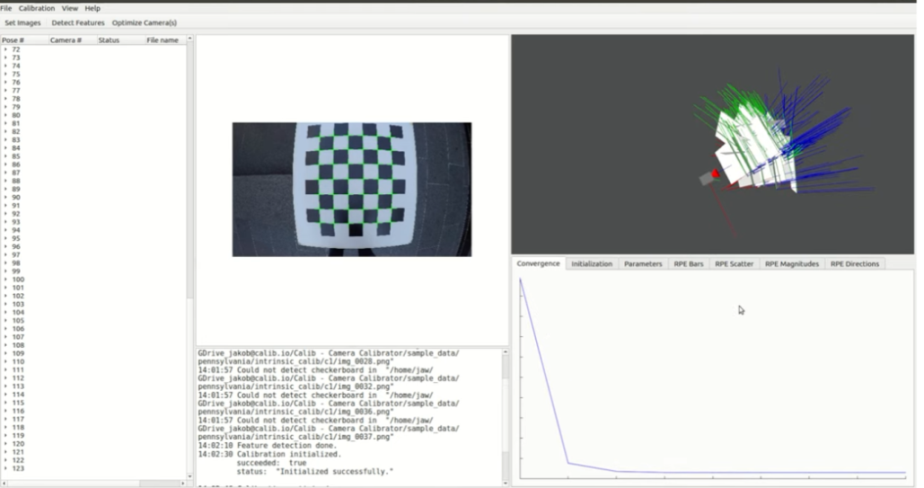
*Рисунок 23 Визуальный пример среднеквадратичной ошибки*

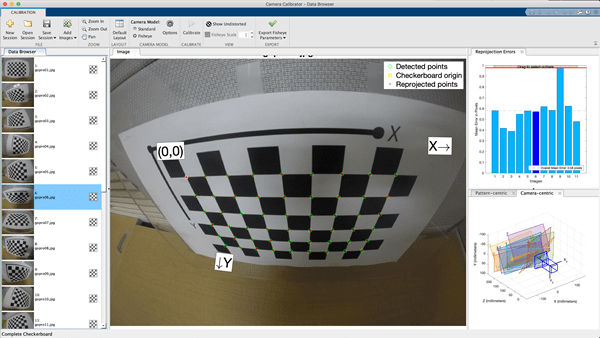
Среднеквадратичная ошибка 1.0 означает, что в среднем каждая из точек детеции находится на расстоянии 1.0 пикселя от своего фактического положения.

# Выявление аналогов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Функциональные критерии | Проект | ПО заказчика | Calib.io | MATLAB | Camera-calib |
| Разные алгоритмы калибровки | + | - | + | + | - |
| Видеопоток камеры в качестве источника изображений | + | + | - | + | + |
| Работа с промышленной камерой Basler | + | + | - | - | - |
| Стерео  калибровка | + | - | + | + | - |

При исследовании аналогов, мною было выделено четыре программы выполняющие калибровку камеры, и проведено исследование по критерию функциональности согласно следующей таблице:

1. Компания Calib.io, производящая высокоточные калибровочные платы, так же имеет собственную программу для калибровки камеры  
   *Рисунок 25 Пользовательский интерфейс программа аналога компании “Calib.io”*
2. **Модуль калибровки камеры для математического программного обеспечения “MATLAB ”**



*Рисунок 26 Пользовательский интерфейс программы “MATLAB”*

Представленные аналоги не могут в полной мере удовлетворить требования заказчика.

# Анализ требований

# Цель разработки

Целью разработки является создание программного обеспечения, способного усовершенствовать результаты калибровки существующего программного обеспечения калибровки видеокамер Заказчика.

В процессе будет разработана программа, удовлетворяющая требования калибровки и предоставляющие функции анализа полученных результатов. В дальнейшем планируется внедрение разработанного программного обеспечения в отдел комплексных разработок Заказчика.

# Функциональные требования

Разработанная программа должна содержать следующие возможности:

1. Загрузка изображений из физического носителя
2. Добавление изображений напрямую с Камеры.
3. Детекция калибровочного шаблона, с указанием параметров, добавленных в программу изображений.
4. Функция просмотра полученных изображений и результатов детекции.
5. Функция выполнения калибровки камеры с использованием калибровочных параметров.
6. Функция просмотра полученных результатов калибровки
7. График анализа выполненной калибровки по каждому изображению
8. Функция просмотра и сравнения нескольких калибровок с разными настройками калибровочного шаблона

# Проектирование

# Выбор языка программирование

Стек технологий заказчика базируется использованием языка С++. Для дальнейшего поддерживания и внедрение программного решения, было принято решение использовать язык С++ стандарта С++17. Для сериализации данных был выбран язык разметки Yaml.

# Выбор среды разработки

В качестве среды разработке был выбран инструмент Qt creator, который основан на языке С++ и фреймворка Qt.   
Фреймворк Qt позволяет создавать пользовательские окна, работать с файловой системы, графиками, таблицами и изображениями.  
Версия Qt 6.0.3, полностью удовлетворяет поставленной задачи проектирования.

Для работы с алгоритмами компьютерного зрения была выбрана библиотека OpenCV версии 4.5.5.

Для работы с промышленными Камера, необходима библиотека Pylon SDK версии 6.3.0.23157.

# Архитектура проекта

При создании программы, использовался объектно-ориентированный подход. Во взаимодействии участвуют следующие классы:

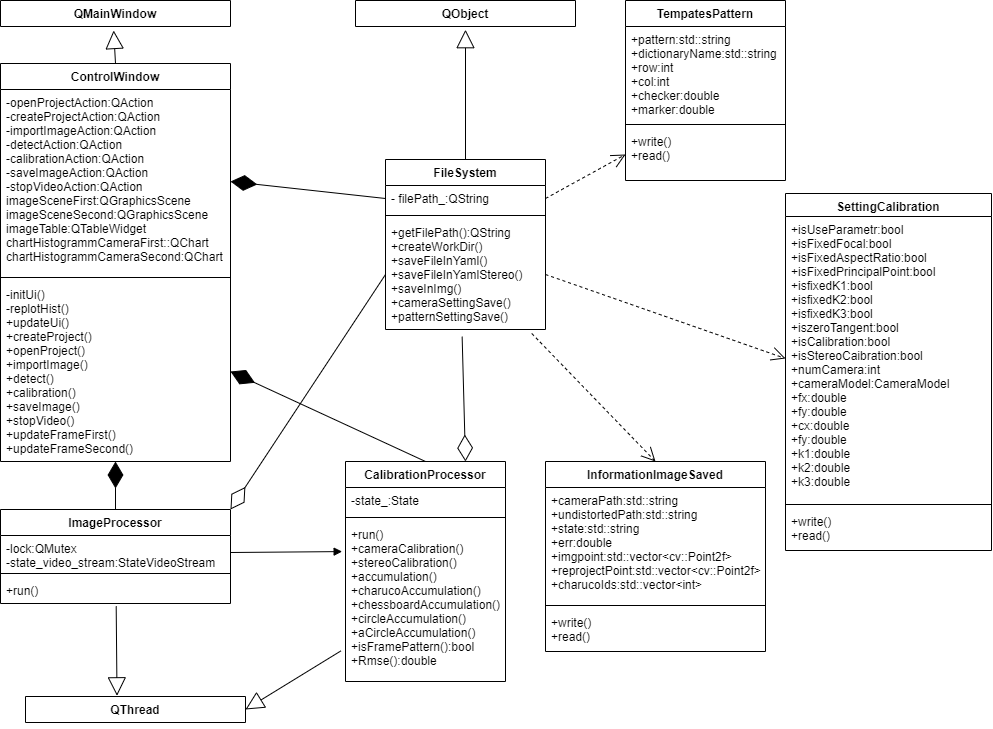
1. Класс ControlWindow – главное пользовательское окно. Содержит в себе следующие виджеты:

* Таблица изображений
* Два графических поля просмотра изображений
* Текстовое поле, отображающее сообщения действий программы
* Настройки шаблона калибровки
* Настройки камеры
* Таблицу сравнения результатов с проектами текущей директории
* Текстовое поле с информацией о настройках текущего проекта
* Текстовое поле вывода результатов по калибровке камеры 1, камеры 2, стереокамеры.
* Столбчатая гистограмма ошибок для камеры 1 и камеры 2

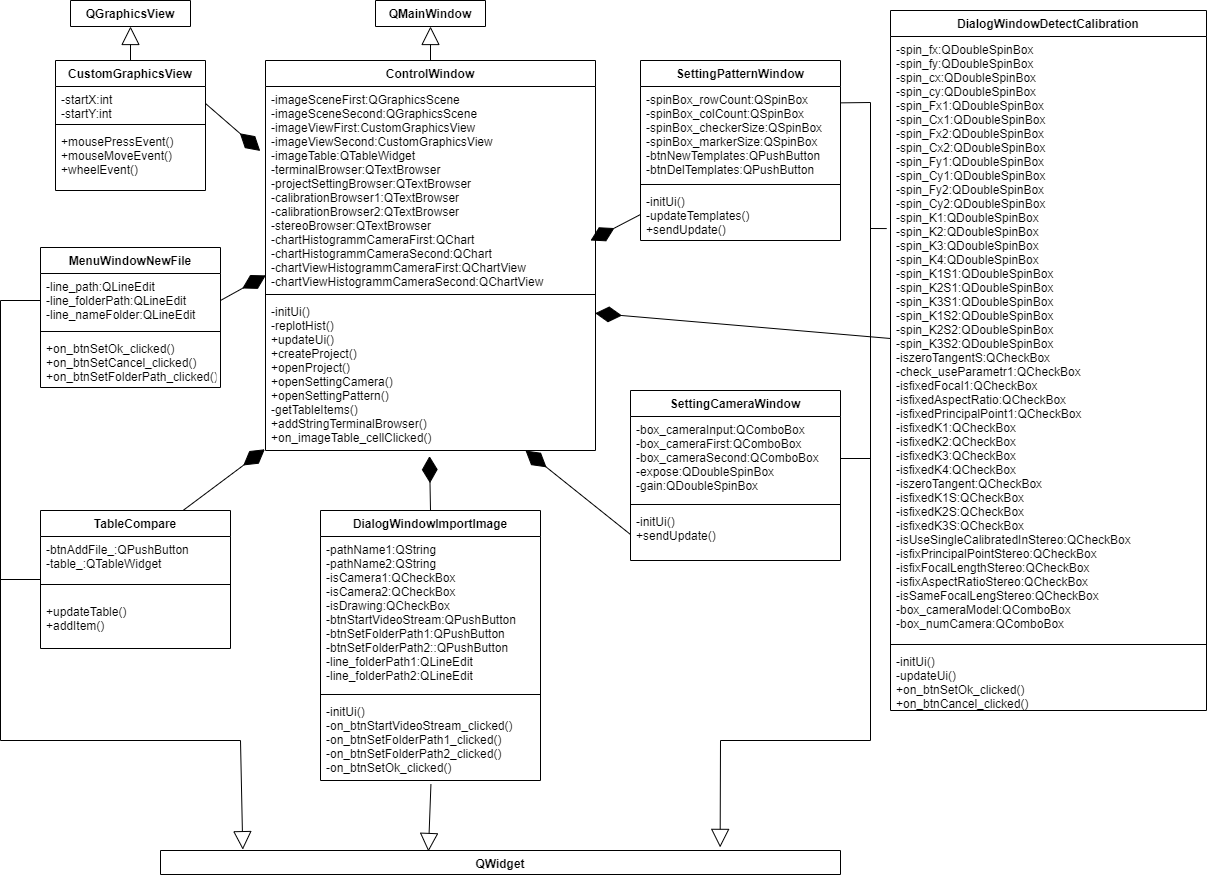
1. Класс ImageProcessor является потоком для взаимодействия между главным пользовательским окном и камерами. Позволяет считывать видеосигналы с камер и отправлять в графическое поле для просмотра.
2. Класс CalibrationProcessor является потоком для детекции углов шаблона и выполняет калибровку камеры.
3. Класс FileSystem занимается чтением/записью информации в файл и из файла. Работает с файловой системы, создаёт и инициализирует структуры данных, такие как:

* SettingCalibration – структура, хранящая в себе настройки для калибровки
* InformationImageSaved – структура, в которой хранится вся информация об изображении
* TemplatesPattern – структура, в которой хранится информация о заранее записанных настройках калибровочного шаблона

1. Класс DialogWindowImportImage является диалоговым окном при нажатии на кнопку загрузки изображений. Позволяет пользователю выбрать источник изображений.
2. Класс DialogWindowDetectCalibration является диалоговым окном при нажатии на кнопку калибровки. Позволяет пользователю выбрать тип калибровки и её настройки.
3. Класс MenuWindowNewFile является диалоговым окном при нажатии на кнопку создания проекта. Позволяет пользователю выбрать директорию и имя для нового проекта.
4. Класс SettingCameraWindow является виджетом для выбора типа и настройки камеры.
5. Класс SettingPatternWindow является виджетом для выбора калибровочного шаблона.
6. Класс TableCompare является виджетом - таблицей для сравнения результатов с другими калибровками.



*Рисунок 27 UML-диаграмма классов*



*Рисунок 28 UML-диаграмма классов пользовательского интерфейса*

# Реализация

# Технические характеристики камеры

В данной дипломной работе используется промышленная камера Basler модели daA1280-54um



*Рисунок 29 Камера Basler daA1280-54um*

*Таблица 2 Характеристики камеры*

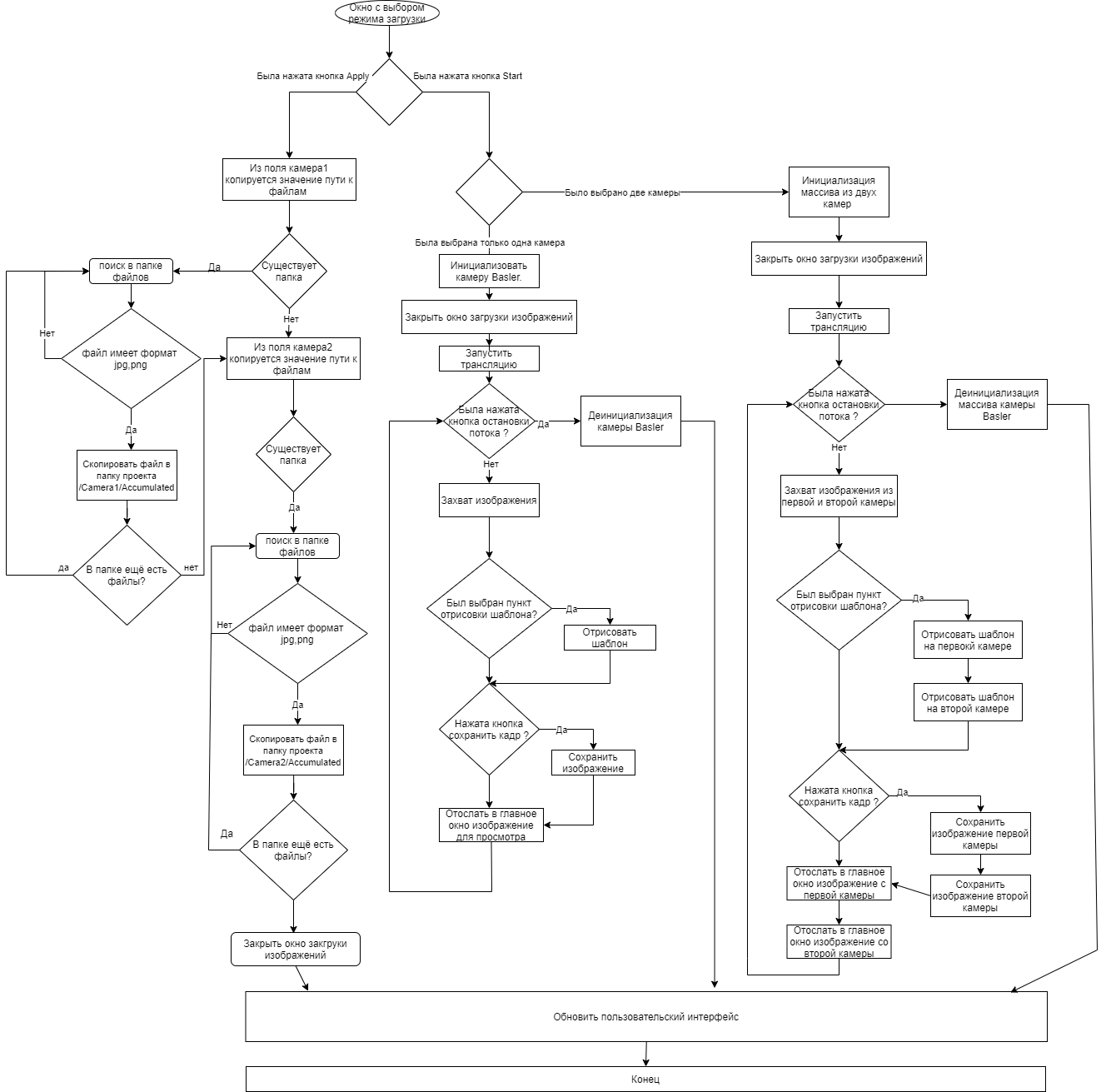
|  |  |
| --- | --- |
| Разрешение камеры | 1280 x 960 |
| Тип сенсора | CMOS |
| Диагональ датчика | 6 mm |
| Размер пикселя | 3.75 μm x 3.75 μm |
| Частота кадров | 54 fps |
| Mono / Color | Mono |
| Интерфейс передачи данных | USB 3.0 |

# Алгоритм загрузки изображений

При нажатии на кнопку Import image в главном пользовательском окне, откроется меню выбора способа загрузки изображения.  
Доступно два режима загрузки

1. Скопировать изображения из директории
2. Запустить трансляцию видеосигнала с камеры

Алгоритм работы представлен на следующей блок-схеме:

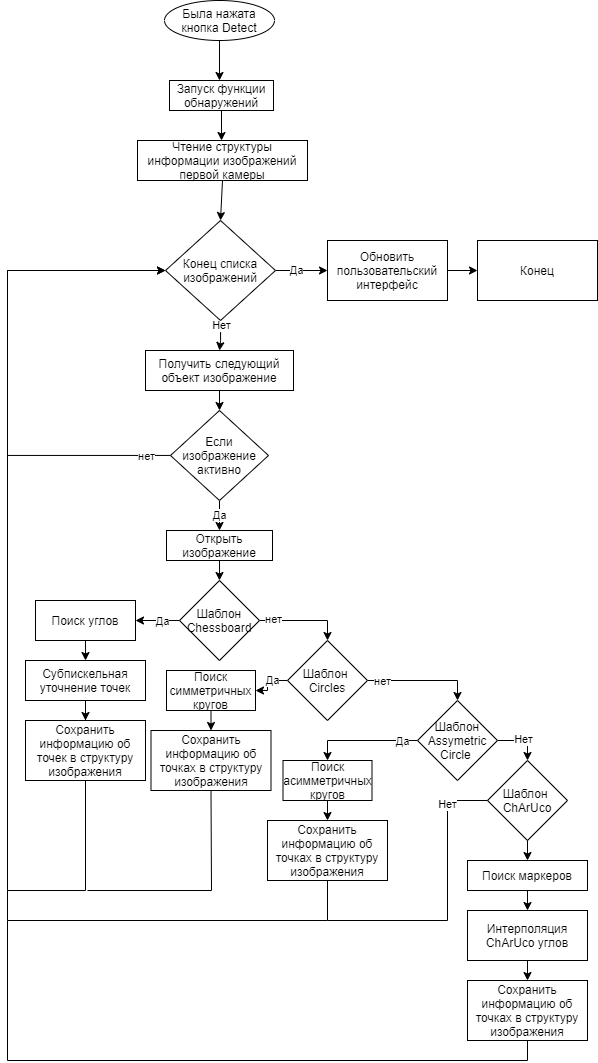


*Рисунок 30 Блок-схема алгоритма загрузки изображений*

# Алгоритм детекции шаблона

При нажатии на кнопку Detect, запускается алгоритм обнаружения углов шаблона.

Алгоритм работы представлен на следующей блок-схеме:



*Рисунок 31 Блок-схема алгоритма детекции*

При детекции шаблона “Шахматная доска”, используется функция   
cv::findChessboardCorners(gray,cv::Size(), corner\_pts);

Где

gray – прочитанное и переведённой в чёрно-белую цветовую палитру изображение

cv::Size() – размер шахматной доски строки на столбцы  
corner\_pts –выходной массив двумерных точек обнаруженных углов  
  
В случае калибровки шахматной доски, дополнительно применяется функция уточнения углов шахматной доски:   
cv::cornerSubPix(gray,corner\_pts, cv::Size(11,11), cv::Size(-1,-1), criteria);  
Где

gray – прочитанное и переведённой в чёрно-белую цветовую палитру изображение

corner\_pts –выходной массив двумерных точек обнаруженных углов

cv::Size(11,11) – размер окна поиска субпиксельной итерации

cv::Size(-1,-1) – Показатель мёртвой области, значение (-1,-1) указывает на её отсутствие  
criteria – критерий завершений итеративного процесса уточнения угла.

При детекции шаблона “Симметричные круги” и “Асимметричные круги” используется функция

cv::findCirclesGrid(gray,cv::Size(),corner\_pts,flag);

Где

gray - прочитанное и переведённой в чёрно-белую цветовую палитру изображение

cv::Size() - размер калибровочного шаблона

corner\_pts - выходной массив двумерных точек обнаруженных центров кругов

flag – флаг,

При детекции симметричных кругов используется

cv::CALIB\_CB\_SYMMETRIC\_GRIDПри детекции асимметричных кругов используется

cv::CALIB\_CB\_ASYMMETRIC\_GRID

Функция определяет сетку кругов во входном изображении. Если круги были найдены, функция определяет их центр.

При детекции шаблона “ChArUco” используется функция

cv::aruco::detectMarkers(gray, charucoboard\_->dictionary, markerCorners, markerIds, params\_);

Где  
gray - прочитанное и переведённой в чёрно-белую цветовую палитру изображение

charucoboard\_->dictionary – тип маркеров, по которым будет выполняться обнаружение

markerCorners – вектор обнаруженных углов маркеров. Для каждого маркера указаны его четыре угла. Для N обнаруженных маркеров размеры этого массива равны Nx4. Порядок расположения углов – по часовой стрелке.

markerIds – вектор идентификаторов обнаруженных маркеров.

Идентификаторы имеют тот же порядок, что и маркеры в массиве markerCorners

params\_ - параметры обнаружения маркеров

После обнаружения маркеров, выполняется интерполяция углов – получение 2D положений углов шахматной доски ChArUco, используя обнаруженные маркеры.  
cv::aruco::interpolateCornersCharuco(markerCorners, markerIds, gray, charucoboard\_, corner\_pts, charucoIds);

Где

charucoboard\_ - размер шаблона

corner\_pts - выходной массив двумерных точек обнаруженных углов шахматной доски

# Алгоритм калибровки

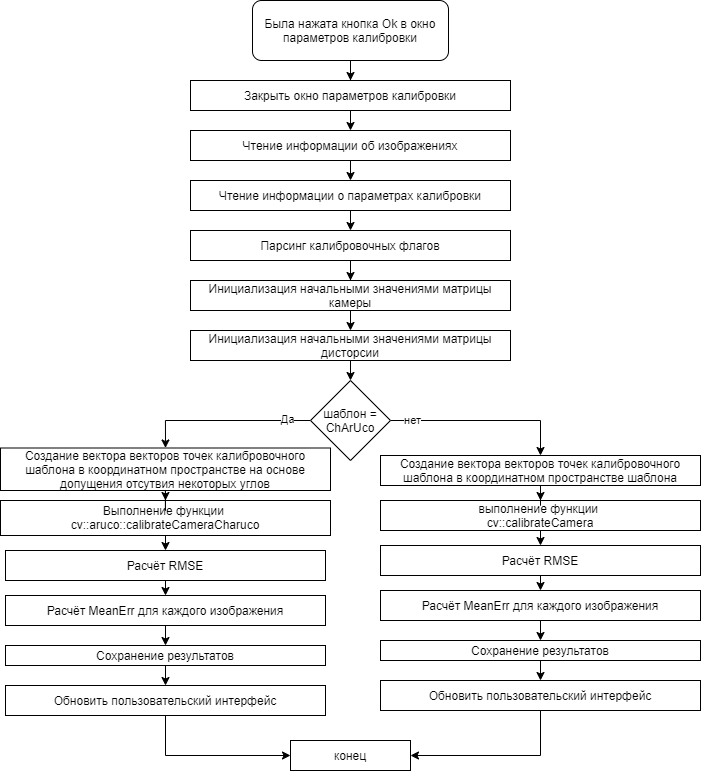
При нажатии на кнопку Calibration открывается диалоговое окно с параметрами калибровки. Пользователь имеет возможность выбрать камеру, которую будет калибровать, ввести, необходимы данные и активировать флаги для калибровки.

При нажатии на кнопку Calibration открывается диалоговое окно с параметрами калибровки.

Во время открытия окна, подгружаются соответствующие значения , ,, , , , для выбранной камеры. Чтения параметров осуществляется из файла формата Yaml в папке Result соответствующей камеры. Значения калибровки парсится и записывается в соответствующие числовые полня. При выбранной камере, чтение параметров меняется на индекс, соответствующей камере. Индекс первой камеры равен единице, индекс второй камеры равен двойки.

При нажатии на кнопку Ok в диалоговом окне параметров калибровки, запуститься алгоритм калибровки камеры. Значения из диалогового окна временно запишутся в структуру SettingCalibration, они будут считаны когда будет проходить инициализация внутренней матрицы камеры и матрицы дисторсии.

Алгоритм работы представлен на следующей блок-схеме:



*Рисунок 32 Блок-схема алгоритма калибровки*

Точки калибровочного шаблона в координатном пространстве при каждом шаблоне создаётся по-разному.

Шахматная доска и симметричные круги имеют следующий алгоритм:

for(int i{0}; i<col; i++)

{

for(int j{0}; j<row; j++)

objp.push\_back(cv::Point3f(j,i,0));

}

Координаты пространства асимметричного шаблона находится следующем образом:

for(int i{0}; i<col; i++)

{

for(int j{0}; j<row; j++)

objp.push\_back(cv::Point3f(2\*j + i%2,i,0));

}

В случае с особенностями шаблона ChArUco, может появиться ситуация, когда координаты пространства могут не совпадать с углами, разный размер этих массивов приведёт к ошибке программы. Для решения этой проблемы был применён следующий алгоритм:

objpoints.resize(markerIds.size());

for(unsigned int i = 0; i < markerIds.size(); i++)

{

unsigned int nCorners = (unsigned int)markerIds[i].size();

objpoints[i].reserve(nCorners);

for(unsigned int j = 0; j < nCorners; j++)

{

int pointId = markerIds[i][j];

objpoints[i].push\_back(charucoboard\_->chessboardCorners[pointId]);

}

}

Калибровку камеры выполняет следующая функция: cv::calibrateCamera(objpoints, imgpoints, cv::Size(img.rows, img.cols),

cameraMatrix, disCoeffs, R, T, calibrationFlags\_);

Где

Objpoints – точки калибровочного шаблона в координатном пространстве шаблона

Imgpoints – точки углов калибровочного шаблона

cv::Size(img.rows, img.cols) – размер калибровочного шаблона

cameraMatrix – матрица камеры

disCoeffs – матрица дисторсии

R – вектор вращения

T – вектор перемещения

calibrationFlags – калибровочные флаги

cv:: CALIB\_USE\_INTRINSIC\_GUESS - допускает начальные значения внутренней матрицы камеры и оптимизирует их.

cv:: CALIB\_FIX\_FOCAL\_LENGTH – фокусное расстояние не изменяется и равно начальным значением

cv:: CALIB\_FIX\_ASPECT\_RATIO – функция рассматривает только в качестве свободного параметра. Соотношение остаётся неизменным.

cv:: CALIB\_FIX\_PRINCIPAL\_POINT – точка не изменяется в результате калибровки, остаётся начальным значением

cv:: CALIB\_ZERO\_TANGENT\_DIST – Радиальная дисторсия ( равна 0 и не подсчитывается

cv:: CALIB\_FIX\_K1 – коэффициент не изменятся во время калибровки и равен начальному значению

cv:: CALIB\_FIX\_K2– коэффициент не изменятся во время калибровки и равен начальному значению

cv:: CALIB\_FIX\_K3– коэффициент не изменятся во время калибровки и равен начальному значению

В случае калибровки шаблона ChArUco используется следующая функция:

cv::aruco::calibrateCameraCharuco(imgpoints, markerIds,

charucoboard\_,cv::Size(img.rows, img.cols),

cameraMatrix,disCoeffs,R, T, calibrationFlags\_);

где

markerIds – массив идентификаторов маркеров

Флаги калибровки такие же как и у функции cv::calibrateCamera()

Для стерео калибровки необходимо прочитать пары изображений и использовать уже откалиброванные параметры камеры 1 и камеры 2.  
Стерео калибровка выполняется функцией:

stereoCalibrate(objectPoints, imagePoints0, imagePoints1,cameraMatrix[0], disCoeffs[0],cameraMatrix[1], disCoeffs[1],imageSize, R, T, E, F,flag,)

Где

objectPoints - точки калибровочного шаблона в координатном пространстве шаблона  
imagePoints0 - точки углов калибровочного шаблона камеры 1

imagePoints1 - точки углов калибровочного шаблона камеры 2

cameraMatrix[0] – внутренняя матрица камеры 1

cameraMatrix[1] – внутренняя матрица камеры 2

disCoeffs[0] – коэффициенты дисторсии камеры 1

disCoeffs[0] – коэффициенты дисторсии камеры 2

imageSize –размер изображения

R – вектор поворота

T – вектор перемещения

E – существенная матрица

F – основополагающая матрица

Flag: калибровочные флаги

cv::CALIB\_SAME\_FOCAL\_LENGTH –

принудительно

cv::CALIB\_FIX\_ASPECT\_RATIO - оптимизация . Фиксирование соотношения

cv::CALIB\_FIX\_FOCAL\_LENGTH – Фиксация и .

cv::CALIB\_FIX\_PRINCIPAL\_POINT – точки не изменяется в результате калибровки, остаётся начальным значением

cv::CALIB\_FIX\_INTRINSIC – Использование откалиброванных параметров каждой камеры.

cv::CALIB\_ZERO\_TANGENT\_DIST - Радиальная дисторсия ( равна 0 и не подсчитывается

cv::CALIB\_FIX\_K1 - коэффициенты не изменятся во время калибровки и равен начальному значению

cv::CALIB\_FIX\_K2 - коэффициенты не изменятся во время калибровки и равен начальному значению

cv::CALIB\_FIX\_K2 - коэффициенты не изменятся во время калибровки и равен начальному значению

Стереоисправления выполняет функция:

stereoRectify(cameraMatrix[0], disCoeffs[0],cameraMatrix[1], disCoeffs[1],imageSize, R, T, R1, R2, P1, P2, Q, 1);

Где

R1 – матрица выпрямления для камеры 1

R2 – матрица выпрямления для камеры 2

P1 – матрица проекции в новых координатах для камеры 1

P2 – матрица проекции в новых координатах для камеры 2

Q – матрица отображения несоответствия

# Расчёт RMSE

Учитывая внутреннюю матрицу, матрицы искажения, вращения и перемещения, мы должны сначала преобразовать точку объекта в точку изображения с помощью projectPoints(). Затем мы можем вычислить абсолютную норму между тем, что мы получили с помощью нашего преобразования, и алгоритмом поиска угла. Чтобы найти среднюю ошибку, мы вычисляем среднее арифметическое ошибок, вычисленных для всех калибровочных изображений.

Алгоритм выглядит следующим образом:

for(size\_t i = 0; i < objpoints.size(); ++i )

{

projectPoints(objpoints[i], R[i], T[i], cameraMatrix, disCoeffs, imagePoints2);

err = norm(imgpoints[i], imagePoints2, cv::*NORM\_L2*);

imageInfo[indexImages[i]].err = err;

meanErr +=err;

imageInfo[indexImages[i]].reprojectPoint = imagePoints2;

size\_t n = objpoints[i].size();

totalErr += err\*err;

totalPoints += n;

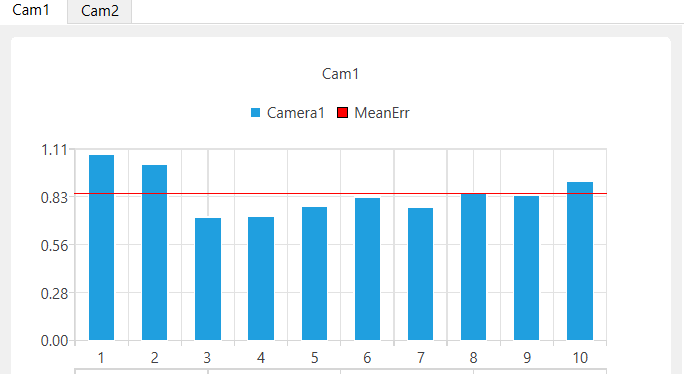
}

meanErr /= objpoints.size();

return std::sqrt(totalErr/totalPoints);

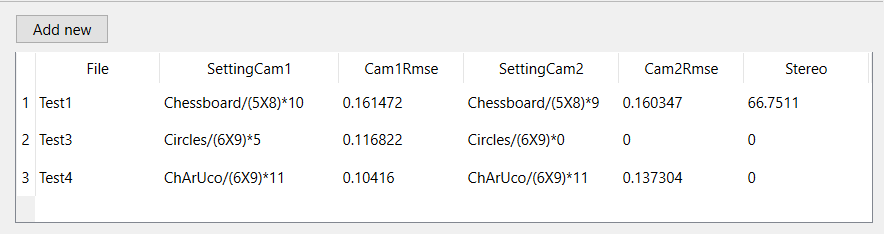
# Инструменты анализа полученных результатов

Для анализа полученных результатов калибровки представлено не только расчётное значение среднеквадратичной ошибки, но и следующие инструменты:

1. Столбчатая гистограмма, по x – номер снимка изображения, по y – ошибка перепроецирования. Горизонтальная линия MeanErr равна среднему значению ошибки перепроецирования. Гистограмма помогает определить, какое изображение даёт большую ошибку, для дальнейшего устранения негативных результатов. При нажатии на столбец гистограммы, на экране откроется соответствующее изображение.

*Рисунок 33 Столбчатая гистограмма для камеры 1*

1. Таблица сравнения результатов, сделанных с разными калибровочными платами. Таблица помогает проанализировать полученные результаты и выявить оптимальный шаблон для калибровки камеры. При нажатии на кнопку Add new, создаётся новый проект в той же директории, что и текущий. Пользователю остаётся ввести только имя проекта.



*Рисунок 34 Таблица сравнения результатов*

1. На графическом экране при открытии изображения, при наличии углов детекции и калибровочных результатов.

Зелёным перекрестием указана точка, обнаруженная алгоритмами детекции.

Красным перекрестием указана точка перепроецирования.

Красная окружность показывает среднеквадратичное отклонение.

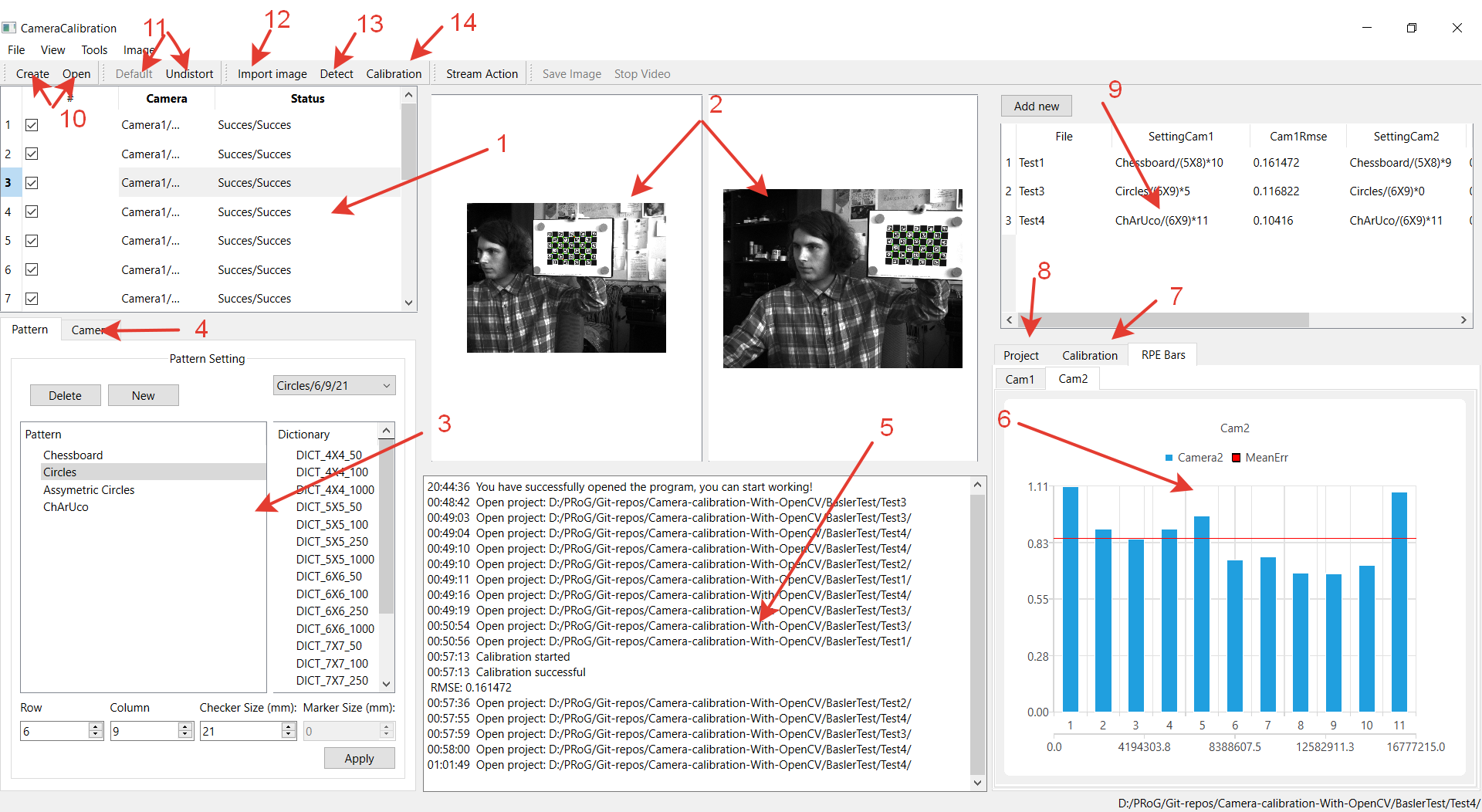
Синяя линия показывает направление и значение (длинна) ошибки перепроецирования для данного изображения.

*Рисунок 35 Пример обозначения угла в графическом поле*

# 2.3.6. Пользовательский интерфейс

Пользовательский интерфейс главного окна предоставляет пользователю следующую информацию:

1. Таблица списка изображений
2. Два графических поля для просмотра изображений
3. Виджет настройки калибровочного шаблона
4. Виджет настройки камеры
5. Текстовое поле с Log информацией
6. Гистограмма ошибок перепроецирования
7. Текстовое поле с результатами калибровки
8. Текстовое поле с настройками проекта
9. Виджет таблицы просмотра калибровок
10. Кнопки создания/открытия проекта
11. Кнопки переключение между просмотром исходных изображение и undistorted изображений
12. Кнопка запуска диалогового окна, начало добавления изображений
13. Кнопка запуска алгоритма детекции
14. Кнопка запуска диалогового окна параметров калибровки



*Рисунок 36 Главное пользовательское окно*

Диалоговое окно для загрузки изображений поделено на две части:

Левая часть отвечает за загрузку изображений из директории, правая часть за трансляцию с камеры.

При нажатии на кнопку “…” открывается проводник, где необходимо выбрать директорию, после выбора, её путь автоматически запишется в соответствующее поле Camera1 или Camera2

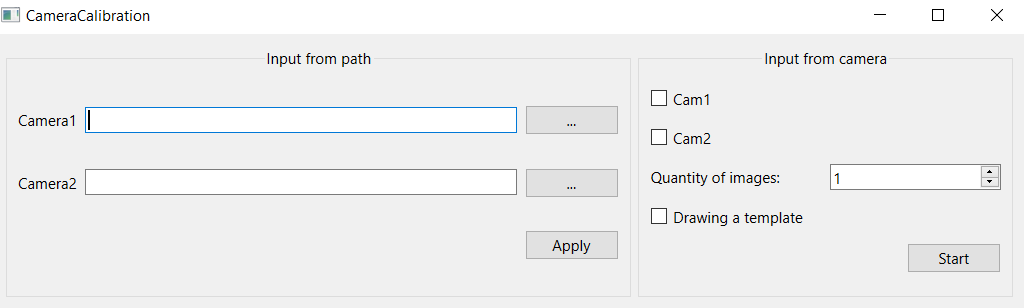
При активации чекбокса Cam1 будет запущена трансляция с первой камеры

При активации чекбокса Cam2 будет запущена трансляция со второй камеры

При одновременной активации чекбокса Cam1 и Cam2 будет запущена трансляция с двух камера

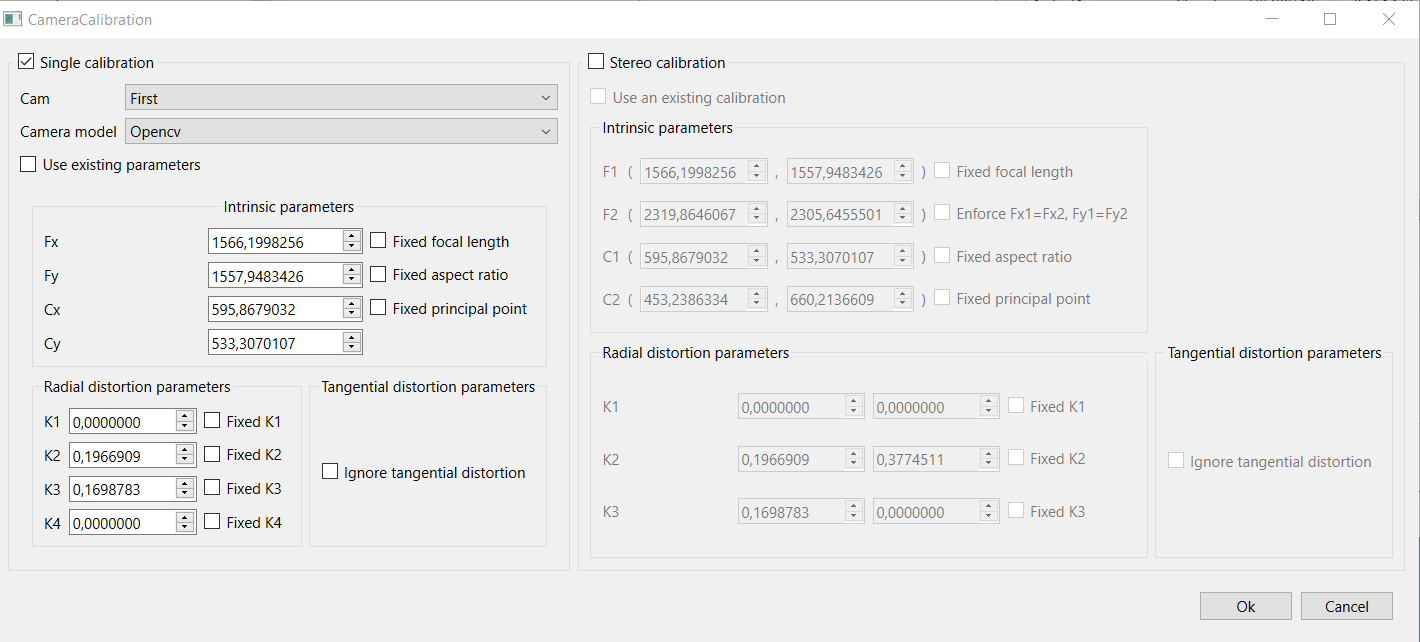
Поле Quantity of image отвечает за кол-во снимков, которые необходимо сделать для завершения трансляции

При активации чекбокса Drawing a template, будет автоматически рисоваться обнаруженный шаблон на входном изображении

Кнопка Apply запускает алгоритм копирования изображений

Кнопка Start запускает алгоритм потока камер

*Рисунок 37 Диалоговое окно загрузки изображений*

Диалоговое окно параметров камеры выглядит следующим образом:

*Рисунок 38 Диалоговое окно параметров калибровки камеры*

Выпадающий список Cam отвечает за выбор камеры, которая будет калиброваться.

Выпадающий список модели камеры, отвечает за выбор модели камеры, на текущий момент существует только одна модель камеры.

Чекбокс Use existing parameters отвечает за калибровочный флаг

cv:: CALIB\_USE\_INTRINSIC\_GUESS

Чекбокс Fixer focal length отвечает за калибровочный флаг

cv:: CALIB\_FIX\_FOCAL\_LENGTH

Чекбокс Fixer aspect ratio отвечает за калибровочный флаг

cv:: CALIB\_FIX\_ASPECT\_RATIO

Чекбокс Fixer principal point отвечает за калибровочный флаг

cv:: CALIB\_FIX\_PRINCIPAL\_POINT

Чекбокс Fixed k1 отвечает за калибровочный флаг

cv:: CALIB\_FIX\_K1

Чекбокс Fixed k2 отвечает за калибровочный флаг

cv:: CALIB\_FIX\_K2

Чекбокс Fixed k3 отвечает за калибровочный флаг

cv:: CALIB\_FIX\_K3

Чекбокс Ignore tangent distortion отвечает за калибровочный флаг

cv:: CALIB\_ZERO\_TANGENT\_DIST

Поля для ввода параметров, значения который соответственно равно начальным значениям при выполнении калибровки

При нажатии на кнопку Ok запускается алгоритм калибровки  
Вторая половина диалогового окна предназначена для стерео калибровки, в настоящий момент она находится в разработке.

# ЗАКЛЮЧЕНИЕ

В результате выполнения дипломного проекта было разработано программное обеспечение для калибровки Камеры, которое закрывает потребность в сравнении и поиске наилучшего алгоритма калибровки камеры.

Разработаны алгоритмы, позволяющие выполнять калибровку камеры, аналитические инструменты, позволяющие анализировать полученные результаты, пользовательский интерфейс.

Работа выполнена согласно требованиям технического задания.

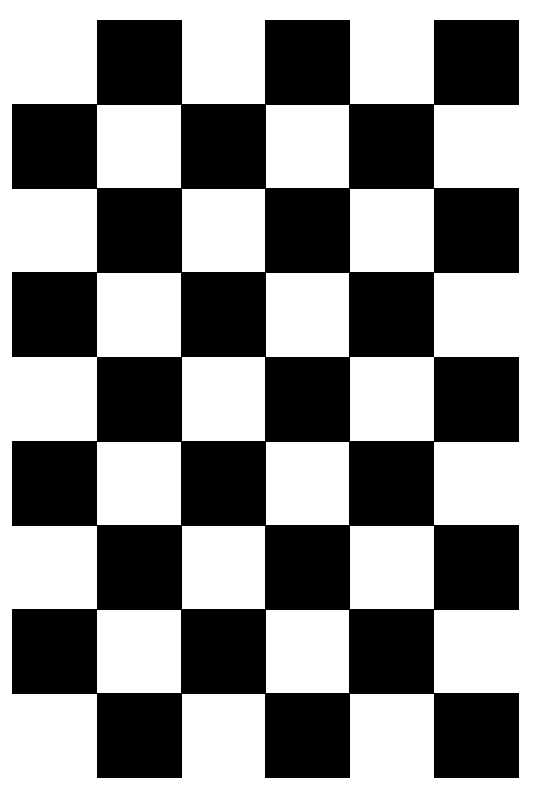
Далее запланирован этап модернизации программы, соответствующие аналитические инструменты, изменение пользовательского интерфейса, более удобного для заказчика и внедрение программного обеспечения в отдел комплексных разработок Заказчика.

В ходе выполнения дипломного проекта исполнитель получил опыт создания программного обеспечения с использованием фреймворка Qt, использования библиотек компьютерного зрения Opencv.

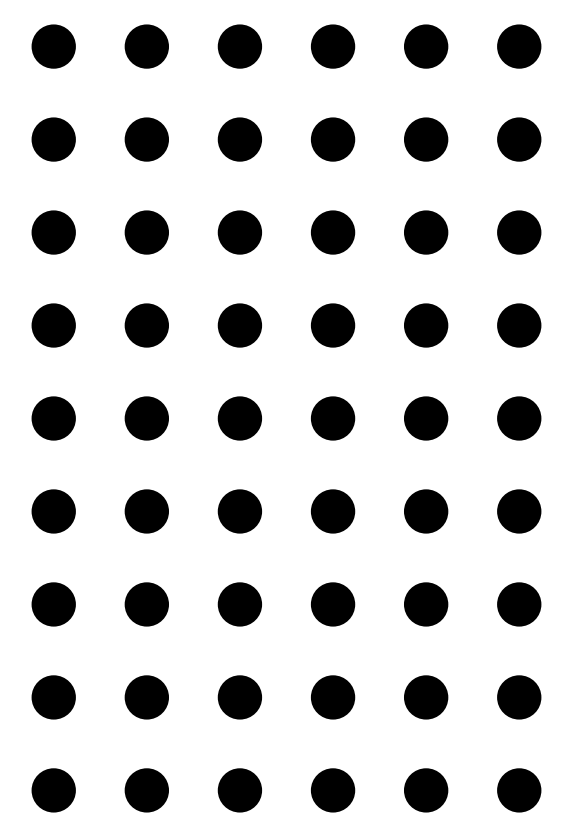
# СПИСОК ЛИТЕРАТУРЫ

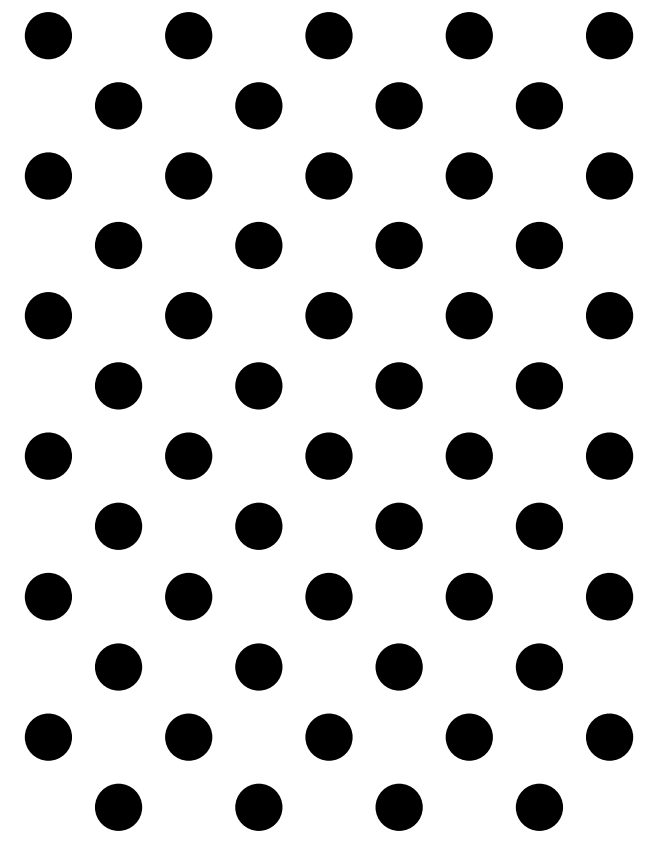
1. Ezust, A. An Introduction to Design Patterns in C++ with Qt A.Ezust, P. Ezust; Editor-in-Chief: Mark Taub Managing Editor: Kristy Hart Project Editor: Jovana San Nicolas-Shirley Copy Editor: Apostrophe Editing Services. – 2nd Edition – 501 Boylston Street, Suite 900 Boston, MA 02116 : y O’Reilly Media, Inc., 2012. – 766 с.
2. Lee, Z. E. Application Development with Qt Creator Z. E. Lee, R. . Rischpater; Commissioning Editor: Richa Tripathi Content Development Editor: Tiksha Sarang Technical Editor: Gaurav Gala Copy Editor: Safis Editing . – Third Edition – Birmingham : Packt Publishing Ltd., 2020. – 413 с.
3. EWBANK, T. Efficient and precise stereoscopic vision for humanoid robots T. EWBANK. – 2016 UNIVERSITY OF LIÈGE, . – 88 с.
4. Barfield, W. Fundamentals Of Wearable Computers and Augmented Reality W. Barfield. – Second edition – Broken Sound Parkway NW : Taylor & Francis Group, LLC, 2016. – 740 с.
5. Learning Image Processing with OpenCV First published – Birmingham B3 2PB, UK : Published by Packt Publishing Ltd., 2015. – 319 с.
6. Kaehler, A. Learning OpenCV 3A. Kaehler, G. Bradski; Editor: Dawn Schanafelt Production Editor: Kristen Brown Copyeditor: Rachel Monaghan Proofreader: James Fraleigh . – First Edition – Gravenstein Highway North, Sebastopol : y O’Reilly Media, Inc., 2017. – 1018 с.
7. Hartley, R. E. Multiple View Geometry in Computer Vision R. E. Hartley, A. Zisserman. – First published – United States of America by Cambridge University Press, New York : Cambridge University Press, 2004. – 673 с.
8. Qt 5 and OpenCV 4 Computer Vision Projects Commissioning Editor: Richa Tripathi Content Development Editor: Digvijay Bagul Senior Editor: Afshaan Khan Technical Editor: Abin Sebastian ; First published – Birmingham B3 2PB, UK : Packt Publishing Ltd., 2019. – 339 с.
9. M. Bleyer and C. Breiteneder, “Stereo Matching - State-of-the-Art and Research Challenges,” in Advanced Topics in Computer Vision. Springer, 2013, pp. 143–179.
10. S. Mattoccia, “Stereo vision: algorithms and applications,” 2011.-201 c.
11. Ødegard, T. O. Stereo vision using local methods for autonomous ferry T. O. Ødegard. – 2020 : Norwegian University of Science and Technology, . – 87 с.
12. Users manual for camera link cameras Version 04: 2015. – 178 с.
13. Users manual for GigE vision cameras Version 13: 2013. – 248 с.
14. Библиотека обработки изображений OpenCV [Электронный ресурс]. – URL: <https://docs.opencv.org> (Дата обращения 30.05.2022)
15. Документация к камере Basler модели daA1280-54um [Электронный ресурс]. – URL: <https://docs.baslerweb.com/daa1280-54um> (Дата обращения 30.05.2022)
16. Документация к фреймворку Qt [Электронный ресурс]. – URL: <https://doc.qt.io> (Дата обращения 30.05.2022)
17. Шапиро Л. Компьютерное зрение: Учеб. пособие для вузов / Л. Шапиро, Дж. Стокман. – Пер. с англ. – БИНОМ. Лаборатория знаний, 2006. – 752 стр.
18. Конструктор калибровочных шаблонов [Электронный ресурс]. – URL: <https://calib.io/pages/camera-calibration-pattern-generator> (Дата обращения 30.05.2022)
19. Потапов, А.С. Распознавание образов и машинное восприятие: общий подход на основе принципа минимальной длины описания / А.С. Потапов. – СПб.: Политехника, 2007. – 548 с.
20. Алексей, П. С. СИСТЕМЫ КОМПЬЮТЕРНОГО ЗРЕНИЯ: Учебное пособие / П. С. Алексей. – СПб : Университет ИТМО, 2016. – 162 с.

# Приложение А – шаблон калибровки “шахматная доска”

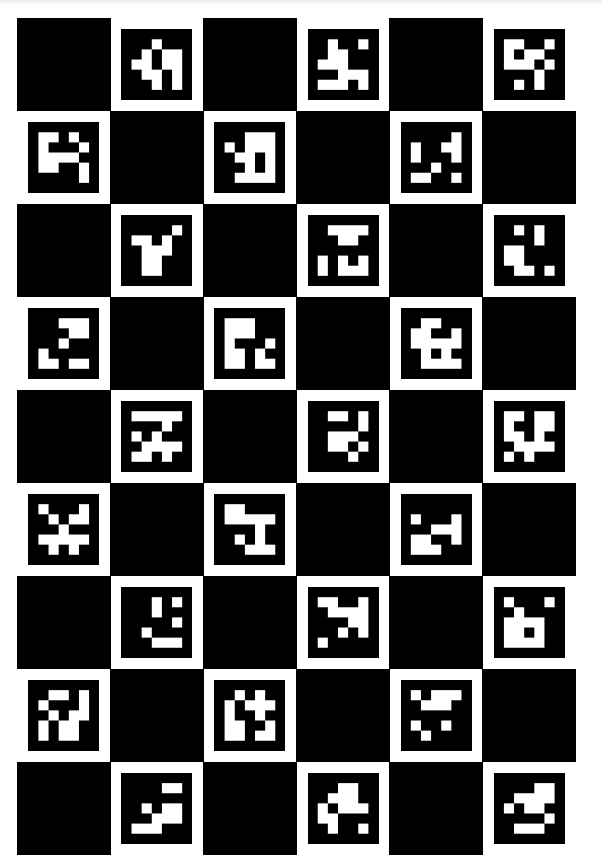
****

# Приложение Б– шаблон калибровки “Круги”



Приложение В– шаблон калибровки “Асимметричные круги” ****

# Приложение Г– шаблон калибровки “ChArUco”

****

# Приложение Д– программный код класса CalibrationProcessor

CalibrationProcessor.h

#ifndef CALIBRATIONPROCESSOR\_H

#define CALIBRATIONPROCESSOR\_H

#include <QString>

#include <QPixmap>

#include <QObject>

#include <QDate>

#include <string>

#include <QMessageBox>

#include <string>

#include <opencv2/opencv.hpp>

#include <opencv2/core/core.hpp>

#include <opencv2/core/types.hpp>

#include <opencv2/core/persistence.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc.hpp>

#include <opencv2/calib3d/calib3d.hpp>

#include <opencv2/aruco.hpp>

#include <opencv2/aruco/charuco.hpp>

#include <opencv2/core/types.hpp>

#include <QThread>

#include "filesystem.h"

class CalibrationProcessor :public QThread

{

Q\_OBJECT

public:

CalibrationProcessor(FileSystem\* fs);

CalibrationProcessor();

enum State{

ACCUMULATION,

CALIBRATION

};

void setState(State state);

void setPattern(QString Pattern);

void setRowCol(int row, int col);

void setMarkerSize(double markerSize);

void setCheckerSize(double checkerSize);

void setDictionaryName(QString dictionaryName);

void setCalibrationFlags(int calibrationFlags);

bool getFrameFromTable(int row);

void reloadVectors();

int translateFlagsOpencv(QString textFlag);

int translateFlagsFisheye(QString textFlag);

void cameraCalibration(std::vector<FileSystem::InformationImageSaved>& imageInfo, FileSystem::SettingCalibration&);

void stereoCalibration(FileSystem::SettingCalibration setting);

void accumulation(std::vector<FileSystem::InformationImageSaved>& imageInfo);

void charucoAccumulation(int i,cv::Mat gray,std::vector<cv::Point2f> corner\_pts, std::vector<FileSystem::InformationImageSaved>& imageInfo);

void chessboardAccumulation(int i,cv::Mat gray,

std::vector<cv::Point2f> corner\_pts,

std::vector<FileSystem::InformationImageSaved>& imageInfo);

void circleAccumulation(int i,cv::Mat gray,

std::vector<cv::Point2f> corner\_pts,

std::vector<FileSystem::InformationImageSaved>& imageInfo);

void aCircleAccumulation(int i,cv::Mat gray,

std::vector<cv::Point2f> corner\_pts,

std::vector<FileSystem::InformationImageSaved>& imageInfo);

bool isFramePattern(cv::Mat\* frame,QString pattern,int row, int col,

double icheckerSize, double imarkerSize, int idictionary);

void createImgUndistorted(std::vector<FileSystem::InformationImageSaved>& imageInfo,

cv::Mat cameraMatrix,cv::Mat disCoeffs,

int numCam);

void saveInImgDrawing(QPixmap qpixmap, QString fileName,int i,

std::vector<FileSystem::InformationImageSaved>& imageInfo);

double Rmse(std::vector<FileSystem::InformationImageSaved>& imageInfo,

std::vector<std::vector<cv::Point2f>> imgpoints,

std::vector<std::vector<cv::Point3f>> objpoints,

cv::Mat cameraMatrix,cv::Mat disCoeffs,

std::vector<cv::Mat> R,std::vector<cv::Mat> T,

std::vector<int> indexImages,double& meanErr);

int getDictionary(QString);

signals:

void sendStatusImg(QString status, int row);

void sendTerminalMessage(QString);

void sendCalibBrowser();

void updateCantrolUi();

public slots:

void run() override;

void setTargetType(QString qstring);

void setTargetSize(int row,int col, double markerSize, double checkerSize, QString dictionaryName);

private:

int succes;

int bad;

bool isRefindStrategy\_;

int calibrationFlags\_;

int CHECKERBOARD\_[2]; //размер шахматной доски убрать автоэкспозицию!

double markerSize\_;

double checkerSize\_;

double rmse\_;

QString targetType\_;

QString dictionaryName\_;

cv::Size imgSizeCharuco\_;

cv::Mat inputFrame\_;

cv::Ptr<cv::aruco::CharucoBoard> charucoboard\_;

cv::Ptr<cv::aruco::Dictionary> dictionary\_;

cv::Ptr<cv::aruco::DetectorParameters> params\_;

std::vector<std::vector<cv::Point2f>> allCharucoCorners;

std::vector<std::vector<int>> allCharucoIds;

std::vector< cv::Mat > allImgs;

FileSystem\* fs\_;

State state\_;

};

#endif // CALIBRATIONPROCESSOR\_H

CalibrationProcessor.cpp

#include "calibrationprocessor.h"

CalibrationProcessor::CalibrationProcessor(FileSystem\* fs)

{

fs\_ = fs;

}

CalibrationProcessor::CalibrationProcessor()

{

}

void CalibrationProcessor::setState(State state)

{

state\_ = state;

}

void CalibrationProcessor::run()

{

targetType\_ = fs\_->getPattern();

CHECKERBOARD\_[0] = fs\_->getRow();

CHECKERBOARD\_[1] = fs\_->getCol();

switch (state\_){

case ACCUMULATION:

{

emit sendTerminalMessage("Corner detection started");

succes = 0;

bad = 0;

std::vector<FileSystem::InformationImageSaved> imageInfo1 = fs\_->getInfoCamera1();

std::vector<FileSystem::InformationImageSaved> imageInfo2 = fs\_->getInfoCamera2();

accumulation(imageInfo1);

fs\_->saveInfoCamera1(imageInfo1);

accumulation(imageInfo2);

fs\_->saveInfoCamera2(imageInfo2);

emit sendTerminalMessage(QString("\n Successful images: %1 \n Unsuccessful images: %2").arg(succes).arg(bad));

emit updateCantrolUi();

break;

}case CALIBRATION:

emit sendTerminalMessage("Calibration started");

FileSystem::SettingCalibration setting = fs\_->getCalibrationSetting();

qDebug()<<"SettingCalibration"<<setting.isfixedK1;

qDebug()<<"SettingCalibration"<<setting.isfixedK2;

if(setting.isCaibration)

{

if(setting.numCamera == 1)

{

fs\_->zeroingCalibInfoCamera1();

std::vector<FileSystem::InformationImageSaved> imageInfo1 = fs\_->getInfoCamera1();

cameraCalibration(imageInfo1, setting);

}else if(setting.numCamera == 2)

{

fs\_->zeroingCalibInfoCamera2();

std::vector<FileSystem::InformationImageSaved> imageInfo2 = fs\_->getInfoCamera2();

cameraCalibration(imageInfo2, setting);

}

emit updateCantrolUi();

}else if(setting.isStereoCaibration)

{

stereoCalibration(setting);

}

break;

}

}

void CalibrationProcessor::cameraCalibration(std::vector<FileSystem::InformationImageSaved>& imageInfo, FileSystem::SettingCalibration& setting)

{

std::vector<std::vector<cv::Point3f>> objpoints;

std::vector<std::vector<int>> markerIds;

std::vector<std::vector<cv::Point2f>> imgpoints;

std::vector<cv::Point3f> objp;

std::vector<int> indexImages;

if(targetType\_ == "Chessboard")

{

for(int i{0}; i<CHECKERBOARD\_[0]; i++)

{

for(int j{0}; j<CHECKERBOARD\_[1]; j++)

objp.push\_back(cv::Point3f(j,i,0));

}

}

if(targetType\_ == "Circles")

{

checkerSize\_ = fs\_->getCheckerSize();

for(int i{0}; i<CHECKERBOARD\_[0]; i++)

{

for(int j{0}; j<CHECKERBOARD\_[1]; j++)

objp.push\_back(cv::Point3f(j,i,0));

}

}

if(targetType\_ == "Assymetric Circles")

{

checkerSize\_ = fs\_->getCheckerSize();

for(int i{0}; i<CHECKERBOARD\_[0]; i++)

{

for(int j{0}; j<CHECKERBOARD\_[1]; j++)

objp.push\_back(cv::Point3f(2\*j + i%2,i,0));

}

}

int countImage = 0;

for(int i = 0;imageInfo.size() > i;i++)

{

if(imageInfo[i].isActive == 1)

{

if(targetType\_ == "ChArUco")

markerIds.push\_back(imageInfo[i].charucoIds);

objpoints.push\_back(objp);

countImage++;

imgpoints.push\_back(imageInfo[i].imgpoint);

indexImages.push\_back(i);

imageInfo[i].isCalib = 1;

}

}

switch (setting.cameraModel)

{

case FileSystem::SettingCalibration::OPENCV:

{

int calibrationFlags = 0;

if(setting.isUseParametr)

calibrationFlags = calibrationFlags | cv::CALIB\_USE\_INTRINSIC\_GUESS;

if(setting.isFixedFocal)

calibrationFlags = calibrationFlags | cv::CALIB\_FIX\_FOCAL\_LENGTH;

if(setting.isFixedAspectRatio)

calibrationFlags = calibrationFlags | cv::CALIB\_FIX\_ASPECT\_RATIO;

if(setting.isFixedPrincipalPoint)

calibrationFlags = calibrationFlags | cv::CALIB\_FIX\_PRINCIPAL\_POINT;

if(setting.iszeroTangent)

calibrationFlags = calibrationFlags | cv::CALIB\_ZERO\_TANGENT\_DIST;

if(setting.isfixedK1)

calibrationFlags = calibrationFlags | cv::CALIB\_FIX\_K1;

if(setting.isfixedK2)

calibrationFlags = calibrationFlags | cv::CALIB\_FIX\_K2;

if(setting.isfixedK3)

calibrationFlags = calibrationFlags | cv::CALIB\_FIX\_K3;

if(setting.isfixedK4)

calibrationFlags = calibrationFlags | cv::CALIB\_FIX\_K4;

cv::Mat cameraMatrix(3, 3, cv::DataType<double>::type);

cameraMatrix.at<double>(0) = setting.fx;

cameraMatrix.at<double>(1) = 0;

cameraMatrix.at<double>(2) = setting.cx;

cameraMatrix.at<double>(3) = 0;

cameraMatrix.at<double>(4) = setting.fy;

cameraMatrix.at<double>(5) = setting.cy;

cameraMatrix.at<double>(6) = 0;

cameraMatrix.at<double>(7) = 0;

cameraMatrix.at<double>(8) = 1;

cv::Mat disCoeffs(1, 5, cv::DataType<double>::type);

disCoeffs.at<double>(0) = setting.k1;//k1

disCoeffs.at<double>(1) = setting.k2;//k2

disCoeffs.at<double>(2) = 0;//p1

disCoeffs.at<double>(3) = 0;//p2

disCoeffs.at<double>(4) = setting.k3;//k3

std::vector<cv::Mat> R, T;

cv::Mat img = cv::imread(static\_cast<std::string>(imageInfo[0].cameraPath));

double rmse;

double meanErr = 0;

if(targetType\_ == "Chessboard" || targetType\_ == "Circles" || targetType\_ == "Assymetric Circles")

{

try{

cv::calibrateCamera(objpoints, imgpoints, cv::Size(img.rows, img.cols),

cameraMatrix, disCoeffs, R, T, calibrationFlags\_);

rmse = Rmse(imageInfo,imgpoints, objpoints, cameraMatrix, disCoeffs,R,T,indexImages,meanErr);

sendTerminalMessage(QString("Calibration successful \n RMSE: %1").arg(rmse));

createImgUndistorted(imageInfo,cameraMatrix, disCoeffs, setting.numCamera);

}catch(const std::exception& ex)

{

sendTerminalMessage("Calibration ERROR: " + QString::fromStdString(ex.what()));

emit updateCantrolUi();

return;

}

}else if(targetType\_ == "ChArUco")

{

checkerSize\_ = fs\_->getCheckerSize();

markerSize\_ = fs\_->getMarkerSize();

dictionaryName\_ = fs\_->getDictionaryName();

dictionary\_ = cv::aruco::getPredefinedDictionary(getDictionary(dictionaryName\_));

charucoboard\_ = cv::aruco::CharucoBoard::create(CHECKERBOARD\_[1], CHECKERBOARD\_[0], checkerSize\_, markerSize\_, dictionary\_);

params\_ = cv::aruco::DetectorParameters::create();

try

{

cv::aruco::calibrateCameraCharuco(imgpoints, markerIds, charucoboard\_,cv::Size(img.rows, img.cols),

cameraMatrix,disCoeffs,R, T, calibrationFlags\_);

objpoints.resize(markerIds.size());

for(unsigned int i = 0; i < markerIds.size(); i++)

{

unsigned int nCorners = (unsigned int)markerIds[i].size();

objpoints[i].reserve(nCorners);

for(unsigned int j = 0; j < nCorners; j++)

{

int pointId = markerIds[i][j];

objpoints[i].push\_back(charucoboard\_->chessboardCorners[pointId]);

}

}

rmse = Rmse(imageInfo,imgpoints, objpoints, cameraMatrix, disCoeffs,R,T,indexImages,meanErr);

sendTerminalMessage(QString("Calibration successful \n RMSE: %1").arg(rmse));

createImgUndistorted(imageInfo,cameraMatrix, disCoeffs, setting.numCamera);

}catch(const std::exception& ex)

{

sendTerminalMessage("Calibration ERROR: " + QString::fromStdString(ex.what()));

emit updateCantrolUi();

return;

}

}

QDateTime date;

date = QDateTime::currentDateTime();

std::string sizePatern;

if(targetType\_ == "Chessboard" || targetType\_ == "Circles" || targetType\_ == "Assymetric Circles")

sizePatern = std::to\_string(CHECKERBOARD\_[0]) + " x " + std::to\_string(CHECKERBOARD\_[1]);

else if(targetType\_ == "ChArUco")

sizePatern = std::to\_string(CHECKERBOARD\_[0]) + " x " + std::to\_string(CHECKERBOARD\_[1]) +

"\n" + "MarkerSize:" + std::to\_string(markerSize\_) + "\n" + "CheckerSize:" + std::to\_string(checkerSize\_) +

"\n" + dictionaryName\_.toStdString();

fs\_->saveFileInYaml(objpoints,imgpoints, cameraMatrix, disCoeffs, R, T, countImage, date.toString("yyyy.dd.M--HH:mm:ss"), rmse,meanErr, targetType\_,

sizePatern, setting.numCamera, cv::Size(img.rows, img.cols));

emit updateCantrolUi();

break;

}

case FileSystem::SettingCalibration::OPENCV\_FISHEYE:

{

break;

}

}

}

void CalibrationProcessor::stereoCalibration(FileSystem::SettingCalibration setting)

{

std::vector<FileSystem::InformationImageSaved> imageInfo1 = fs\_->getInfoCamera1();

std::vector<FileSystem::InformationImageSaved> imageInfo2 = fs\_->getInfoCamera2();

//поиск калибровочных углов у двух идентичных изображениях

std::vector<std::vector<cv::Point2f>> imagePoints0;

std::vector<std::vector<cv::Point2f>> imagePoints1;

int countImg = 0, size\_int=0;

if(imageInfo1.size() >= imageInfo2.size())

size\_int = imageInfo2.size();

else if(imageInfo1.size() <= imageInfo2.size())

size\_int = imageInfo1.size();

for(int i = 0;i<size\_int;i++)

{

if(imageInfo1[i].isActive && imageInfo2[i].isActive)

{

countImg++;

imagePoints0.push\_back(imageInfo1[i].imgpoint);

imagePoints1.push\_back(imageInfo2[i].imgpoint);

if(imageInfo1[i].imgpoint.size() != imageInfo2[i].imgpoint.size())

{

sendTerminalMessage(QString("The number of detected points must be equal"));

return;

}

}

}

std::vector<cv::Point3f> objp;

std::vector<std::vector<cv::Point3f>> objectPoints;

double squareSize = checkerSize\_ = fs\_->getCheckerSize();

if(targetType\_ == "Assymetric Circles")

{

checkerSize\_ = fs\_->getCheckerSize();

for(int i{0}; i<CHECKERBOARD\_[0]; i++)

{

for(int j{0}; j<CHECKERBOARD\_[1]; j++)

objp.push\_back(cv::Point3f(2\*j + i%2,i,0));

}

}else

{

for(int j = 0; j < CHECKERBOARD\_[0]; j++ )

for(int k = 0; k < CHECKERBOARD\_[1]; k++ )

objp.push\_back(cv::Point3f(k\*squareSize, j\*squareSize, 0));

}

for(int i = 0; i < countImg; i++)

{

objectPoints.push\_back(objp);

}

cv::Size imageSize ;

cv::Mat img1 = cv::imread(imageInfo1[0].cameraPath);

imageSize=img1.size();

//на этом этапе надо порешать с флагами

cv::Mat R,T;

cv::Mat E, F;

int flag = 0;

if(setting.isSameFocalLengStereo)

flag |= cv::CALIB\_SAME\_FOCAL\_LENGTH;

if(setting.isfixAspectRatioStereo)

flag |= cv::CALIB\_FIX\_ASPECT\_RATIO;

if(setting.isfixFocalLengthStereo)

flag |= cv::CALIB\_FIX\_FOCAL\_LENGTH;

if(setting.isfixPrincipalPointStereo)

flag |= cv::CALIB\_FIX\_PRINCIPAL\_POINT;

if(setting.isUseSingleCalibratedInStereo)

flag |= cv::CALIB\_FIX\_INTRINSIC;

if(setting.iszeroTangentS)

flag |= cv::CALIB\_ZERO\_TANGENT\_DIST;

if(setting.isfixedK1S)

flag |= cv::CALIB\_FIX\_K1;

if(setting.isfixedK2S)

flag |= cv::CALIB\_FIX\_K2;

if(setting.isfixedK3S)

flag |= cv::CALIB\_FIX\_K2;

cv::Mat cameraMatrix[2];

cv::Mat disCoeffs[2];

cv::Mat camMatrix1(3, 3, cv::DataType<double>::type);

camMatrix1.at<double>(0) = setting.fx1;

camMatrix1.at<double>(1) = 0;

camMatrix1.at<double>(2) = setting.cx1;

camMatrix1.at<double>(3) = 0;

camMatrix1.at<double>(4) = setting.fy1;

camMatrix1.at<double>(5) = setting.cy1;

camMatrix1.at<double>(6) = 0;

camMatrix1.at<double>(7) = 0;

camMatrix1.at<double>(8) = 1;

camMatrix1.copyTo(cameraMatrix[0]);

cv::Mat dist1(1, 5, cv::DataType<double>::type);

dist1.at<double>(0) = setting.k1S1;//k1

dist1.at<double>(1) = setting.k2S1;//k2

dist1.at<double>(2) = 0;//p1

dist1.at<double>(3) = 0;//p2

dist1.at<double>(4) = setting.k3S1;//k3

dist1.copyTo(disCoeffs[0]);

cv::Mat camMatrix2(3, 3, cv::DataType<double>::type);

camMatrix2.at<double>(0) = setting.fx2;

camMatrix2.at<double>(1) = 0;

camMatrix2.at<double>(2) = setting.cx2;

camMatrix2.at<double>(3) = 0;

camMatrix2.at<double>(4) = setting.fy2;

camMatrix2.at<double>(5) = setting.cy2;

camMatrix2.at<double>(6) = 0;

camMatrix2.at<double>(7) = 0;

camMatrix2.at<double>(8) = 1;

camMatrix2.copyTo(cameraMatrix[1]);

cv::Mat dist2(1, 5, cv::DataType<double>::type);

dist2.at<double>(0) = setting.k1S2;//k1

dist2.at<double>(1) = setting.k2S2;//k2

dist2.at<double>(2) = 0;//p1

dist2.at<double>(3) = 0;//p2

dist2.at<double>(4) = setting.k3S2;//k3

dist2.copyTo(disCoeffs[1]);

double rms;

//добавить флаги

try{

rms = stereoCalibrate(objectPoints, imagePoints0, imagePoints1,

cameraMatrix[0], disCoeffs[0],

cameraMatrix[1], disCoeffs[1],

imageSize, R, T, E, F,flag,

cv::TermCriteria(cv::TermCriteria::MAX\_ITER + cv::TermCriteria::EPS, 30, 1e-6));

}catch(const std::exception& ex)

{

sendTerminalMessage("Stereo calibration ERROR: " + QString::fromStdString(ex.what()));

emit updateCantrolUi();

return;

}

// CALIBRATION QUALITY CHECK

// because the output fundamental matrix implicitly

// includes all the output information,

// we can check the quality of calibration using the

// epipolar geometry constraint: m2^t\*F\*m1=0

double averageReprojectionErr;

double err = 0;

int npoints = 0;

std::vector<cv::Vec3f> lines[2];

for(int i = 0;i < countImg;i++)

{

int npt = (int)imagePoints0[i].size();

cv::Mat imgpt[2];

for( int k = 0; k < 2; k++ )

{

if(k==0)

imgpt[k] = cv::Mat(imagePoints0[i]);

else

imgpt[k] = cv::Mat(imagePoints1[i]);

cv::undistortPoints(imgpt[k], imgpt[k], cameraMatrix[k], disCoeffs[k], cv::Mat(), cameraMatrix[k]);

cv::computeCorrespondEpilines(imgpt[k], k+1, F, lines[k]);

}

for(int j = 0; j < npt; j++ )

{

double errij = fabs(imagePoints0[i][j].x\*lines[1][j][0] +

imagePoints0[i][j].y\*lines[1][j][1] + lines[1][j][2]) +

fabs(imagePoints1[i][j].x\*lines[0][j][0] +

imagePoints1[i][j].y\*lines[0][j][1] + lines[0][j][2]);

err += errij;

}

npoints += npt;

}

averageReprojectionErr = err/npoints;

cv::Mat R1, R2, P1, P2, Q;

cv::Rect validRoi[2];

cv::stereoRectify(cameraMatrix[0], disCoeffs[0],

cameraMatrix[1], disCoeffs[1],

imageSize, R, T, R1, R2, P1, P2, Q, 1);

QDateTime date;

date = QDateTime::currentDateTime();

std::string sizePatern;

if(targetType\_ == "Chessboard" || targetType\_ == "Circles" || targetType\_ == "Assymetric Circles")

sizePatern = std::to\_string(CHECKERBOARD\_[0]) + " x " + std::to\_string(CHECKERBOARD\_[1]);

else if(targetType\_ == "ChArUco")

sizePatern = std::to\_string(CHECKERBOARD\_[0]) + " x " + std::to\_string(CHECKERBOARD\_[1]) +

"\n" + "MarkerSize:" + std::to\_string(markerSize\_) + "\n" + "CheckerSize:" + std::to\_string(checkerSize\_) +

"\n" + dictionaryName\_.toStdString();

fs\_->saveFileInYamlStereo(cameraMatrix[0],cameraMatrix[1], disCoeffs[0],disCoeffs[1], R, T, E, F, R1, R2, P1, P2, Q,

imageSize,rms, averageReprojectionErr,date.toString("yyyy.dd.M--HH:mm:ss"),

targetType\_, sizePatern, countImg);

sendTerminalMessage(QString("Calibration successful \n RMSE: %1").arg(rms));

}

void CalibrationProcessor::accumulation(std::vector<FileSystem::InformationImageSaved>& imageInfo)

{

std::vector<cv::Point2f> corner\_pts; // Вектор для хранения пикселей координат углов шахматной доски

if(targetType\_ == "ChArUco")

{

checkerSize\_ = fs\_->getCheckerSize();

markerSize\_ = fs\_->getMarkerSize();

dictionaryName\_ = fs\_->getDictionaryName();

dictionary\_ = cv::aruco::getPredefinedDictionary(getDictionary(dictionaryName\_));

charucoboard\_ = cv::aruco::CharucoBoard::create(CHECKERBOARD\_[1], CHECKERBOARD\_[0], checkerSize\_, markerSize\_, dictionary\_);

params\_ = cv::aruco::DetectorParameters::create();

}

cv::Mat gray;

for(int i = 0;i<imageInfo.size();i++)

{

inputFrame\_ = cv::imread(static\_cast<std::string>(imageInfo[i].cameraPath));

if(inputFrame\_.channels() == 3)

cv::cvtColor(inputFrame\_,gray,cv::COLOR\_BGR2GRAY);

else

gray = inputFrame\_;

if(targetType\_ == "Chessboard")

chessboardAccumulation(i, gray,corner\_pts,imageInfo);

if(targetType\_ == "Circles")

circleAccumulation(i, gray,corner\_pts,imageInfo);

if(targetType\_ == "Assymetric Circles")

aCircleAccumulation(i, gray,corner\_pts,imageInfo);

if(targetType\_ == "ChArUco")

charucoAccumulation(i,gray,corner\_pts,imageInfo);

}

}

void CalibrationProcessor::charucoAccumulation(int i,cv::Mat gray,std::vector<cv::Point2f> corner\_pts, std::vector<FileSystem::InformationImageSaved>& imageInfo)

{

std::vector<int> markerIds;

std::vector<std::vector<cv::Point2f>> markerCorners;

cv::aruco::detectMarkers(gray, charucoboard\_->dictionary, markerCorners, markerIds, params\_);

if (markerIds.size() > 0) {

std::vector<int> charucoIds;

cv::aruco::interpolateCornersCharuco(markerCorners, markerIds, gray, charucoboard\_, corner\_pts, charucoIds);

imageInfo[i].imgpoint = corner\_pts;

imageInfo[i].charucoIds = charucoIds;

imageInfo[i].state = "Succes";

succes++;

}else

{

bad++;

imageInfo[i].state = "NoFind";

imageInfo[i].isActive = 0;

}

}

void CalibrationProcessor::chessboardAccumulation(int i,cv::Mat gray,std::vector<cv::Point2f> corner\_pts,

std::vector<FileSystem::InformationImageSaved>& imageInfo)

{

bool isSuccess = false;

isSuccess = cv::findChessboardCorners(gray,cv::Size(CHECKERBOARD\_[1], CHECKERBOARD\_[0]), corner\_pts);

if(isSuccess)

{

cv::TermCriteria criteria(cv::TermCriteria::EPS | cv::TermCriteria::MAX\_ITER, 30, 0.0001);

cv::cornerSubPix(gray,corner\_pts, cv::Size(11,11), cv::Size(-1,-1), criteria);

imageInfo[i].imgpoint = corner\_pts;

imageInfo[i].state = "Succes";

succes++;

}else

{

bad++;

imageInfo[i].state= "NoFind";

imageInfo[i].isActive = 0;

}

}

void CalibrationProcessor::circleAccumulation(int i,cv::Mat gray,std::vector<cv::Point2f> corner\_pts,

std::vector<FileSystem::InformationImageSaved>& imageInfo)

{

bool isSuccess = false;

isSuccess = cv::findCirclesGrid(gray, cv::Size(CHECKERBOARD\_[1], CHECKERBOARD\_[0]), corner\_pts,

cv::CALIB\_CB\_SYMMETRIC\_GRID, cv::SimpleBlobDetector::create());

if(isSuccess)

{

imageInfo[i].imgpoint = corner\_pts;

imageInfo[i].state = "Succes";

succes++;

}else

{

bad++;

imageInfo[i].state= "NoFind";

imageInfo[i].isActive = 0;

}

}

void CalibrationProcessor::aCircleAccumulation(int i,cv::Mat gray,std::vector<cv::Point2f> corner\_pts,

std::vector<FileSystem::InformationImageSaved>& imageInfo)

{

bool isSuccess = false;

isSuccess = cv::findCirclesGrid(gray, cv::Size(CHECKERBOARD\_[1], CHECKERBOARD\_[0]), corner\_pts,

cv::CALIB\_CB\_ASYMMETRIC\_GRID, cv::SimpleBlobDetector::create());

if(isSuccess)

{

imageInfo[i].imgpoint = corner\_pts;

imageInfo[i].state = "Succes";

succes++;

}else

{

imageInfo[i].state= "NoFind";

imageInfo[i].isActive = 0;

bad++;

}

}

int CalibrationProcessor::translateFlagsOpencv(QString textFlag)

{

if(textFlag == "CV\_CALIB\_USE\_INTRINSIC\_GUESS") return 0x00001;

if(textFlag == "CV\_CALIB\_FIX\_ASPECT\_RATIO") return 0x00002;

if(textFlag == "CV\_CALIB\_FIX\_PRINCIPAL\_POINT") return 0x00004;

if(textFlag == "CV\_CALIB\_ZERO\_TANGENT\_DIST") return 0x00008;

if(textFlag == "CV\_CALIB\_FIX\_FOCAL\_LENGTH") return 0x00010;

if(textFlag == "CV\_CALIB\_FIX\_K1") return 0x00020;

if(textFlag == "CV\_CALIB\_FIX\_K2") return 0x00040;

if(textFlag == "CV\_CALIB\_FIX\_K3") return 0x00080;

if(textFlag == "CV\_CALIB\_FIX\_K4") return 0x00800;

else return 0;

}

void CalibrationProcessor::setPattern(QString Pattern)

{

targetType\_ = Pattern;

}

void CalibrationProcessor::setRowCol(int row, int col)

{

CHECKERBOARD\_[0] = row; //y

CHECKERBOARD\_[1] = col; //x

}

void CalibrationProcessor::setMarkerSize(double markerSize)

{

markerSize\_ = markerSize;

}

void CalibrationProcessor::setCheckerSize(double checkerSize)

{

checkerSize\_ = checkerSize;

}

void CalibrationProcessor::setDictionaryName(QString dictionaryName)

{

dictionaryName\_ = dictionaryName;

}

void CalibrationProcessor::setCalibrationFlags(int calibrationFlags)

{

calibrationFlags\_ = calibrationFlags;

}

double CalibrationProcessor::Rmse(std::vector<FileSystem::InformationImageSaved>& imageInfo,

std::vector<std::vector<cv::Point2f>> imgpoints,

std::vector<std::vector<cv::Point3f>> objpoints,

cv::Mat cameraMatrix,cv::Mat disCoeffs,

std::vector<cv::Mat> R,std::vector<cv::Mat> T,

std::vector<int> indexImages,double& meanErr)

{

std::vector<cv::Point2f> imagePoints2;

double totalErr = 0;

double err;

size\_t totalPoints=0;

for(size\_t i = 0; i < objpoints.size(); ++i )

{

projectPoints(objpoints[i], R[i], T[i], cameraMatrix, disCoeffs, imagePoints2);

err = norm(imgpoints[i], imagePoints2, cv::NORM\_L2);

imageInfo[indexImages[i]].err = err;

meanErr +=err;

imageInfo[indexImages[i]].reprojectPoint = imagePoints2;

size\_t n = objpoints[i].size();

totalErr += err\*err;

totalPoints += n;

}

meanErr /= objpoints.size();

return std::sqrt(totalErr/totalPoints);

}

int CalibrationProcessor::getDictionary(QString dictionaryName)

{

if(dictionaryName == "DICT\_4X4\_50") return 0;

else if(dictionaryName == "DICT\_4X4\_50") return 1;

else if(dictionaryName == "DICT\_4X4\_100") return 2;

else if(dictionaryName == "DICT\_4X4\_1000") return 3;

else if(dictionaryName == "DICT\_5X5\_50") return 4;

else if(dictionaryName == "DICT\_5X5\_100") return 5;

else if(dictionaryName == "DICT\_5X5\_250") return 6;

else if(dictionaryName == "DICT\_5X5\_1000") return 7;

else if(dictionaryName == "DICT\_6X6\_50") return 8;

else if(dictionaryName == "DICT\_6X6\_100") return 9;

else if(dictionaryName == "DICT\_6X6\_250") return 10;

else if(dictionaryName == "DICT\_6X6\_1000") return 11;

else if(dictionaryName == "DICT\_7X7\_50") return 12;

else if(dictionaryName == "DICT\_7X7\_100") return 13;

else if(dictionaryName == "DICT\_7X7\_250") return 14;

else if(dictionaryName == "DICT\_7X7\_1000") return 15;

else if(dictionaryName == "DICT\_ARUCO\_ORIGINAL") return 16;

else if(dictionaryName == "DICT\_APRILTAG\_16h5") return 17;

else if(dictionaryName == "DICT\_APRILTAG\_25h9") return 18;

else if(dictionaryName == "DICT\_APRILTAG\_36h10") return 19;

else if(dictionaryName == "NULL") return -1;

}

bool CalibrationProcessor::isFramePattern(cv::Mat\* frame, QString pattern, int row, int col,

double icheckerSize, double imarkerSize, int idictionary)

{

std::vector<cv::Point2f> corner\_pts;

if(pattern == "Chessboard")

{

if(cv::findChessboardCorners(\*frame, cv::Size(col,row), corner\_pts, cv::CALIB\_CB\_FAST\_CHECK))

{

cv::drawChessboardCorners(\*frame, cv::Size(col, row), corner\_pts, true);

return true;

}

}

if(pattern == "Circles")

{

if(cv::findCirclesGrid(\*frame, cv::Size(col,row), corner\_pts, cv::CALIB\_CB\_SYMMETRIC\_GRID, cv::SimpleBlobDetector::create()))

{

cv::drawChessboardCorners(\*frame, cv::Size(col,row), cv::Mat(corner\_pts), true);

return true;

}

}

if(pattern == "Assymetric Circles")

{

if(cv::findCirclesGrid(\*frame, cv::Size(col,row), corner\_pts, cv::CALIB\_CB\_ASYMMETRIC\_GRID, cv::SimpleBlobDetector::create()))

{

cv::drawChessboardCorners(\*frame, cv::Size(col,row), cv::Mat(corner\_pts), true);

return true;

}

}

if(pattern == "ChArUco")

{

cv::Ptr<cv::aruco::Dictionary> dictionary = cv::aruco::getPredefinedDictionary(idictionary);

cv::Ptr<cv::aruco::CharucoBoard> board = cv::aruco::CharucoBoard::create(col,row, icheckerSize, imarkerSize, dictionary);

std::vector<int> markerIds;

std::vector<std::vector<cv::Point2f>> markerCorners;

cv::Ptr<cv::aruco::DetectorParameters> params = cv::aruco::DetectorParameters::create();

cv::aruco::detectMarkers(\*frame, board->dictionary, markerCorners, markerIds, params);

if (markerIds.size() > 0) {

cv::Mat currentCharucoCorners, currentCharucoIds;

cv::aruco::interpolateCornersCharuco(markerCorners, markerIds, \*frame, board, currentCharucoCorners,currentCharucoIds);

cv::aruco::drawDetectedMarkers(\*frame, markerCorners, markerIds);

if(currentCharucoCorners.total() > 0)

cv::aruco::drawDetectedCornersCharuco(\*frame, currentCharucoCorners);

std::vector<cv::Point2f> charucoCorners;

std::vector<int> charucoIds;

cv::aruco::interpolateCornersCharuco(markerCorners, markerIds, \*frame, board, charucoCorners, charucoIds);

//if at least one charuco corner detected

// if (charucoIds.size() > 0)

// cv::aruco::drawDetectedCornersCharuco(\*frame, charucoCorners, charucoIds, cv::Scalar(255, 0, 0));

return true;

}

}

return false;

}

void CalibrationProcessor::createImgUndistorted(std::vector<FileSystem::InformationImageSaved>& imageInfo,

cv::Mat cameraMatrix,cv::Mat disCoeffs,

int numCam)

{

cv::Mat map1,map2, frame, undist;

frame = cv::imread(static\_cast<std::string>(imageInfo[0].cameraPath));

initUndistortRectifyMap(cameraMatrix, disCoeffs,cv::Mat(),

getOptimalNewCameraMatrix(cameraMatrix, disCoeffs, frame.size(), 1, frame.size(), 0),

frame.size(),CV\_16SC2,map1,map2);

for(int i = 0; i < imageInfo.size(); i++){

frame = cv::imread(static\_cast<std::string>(imageInfo[i].cameraPath));

for(int j = 0; j <= frame.rows; j += frame.rows/5)

line(frame, cv::Point(0, j), cv::Point(frame.cols, j), cv::Scalar(0, 255, 0), 2, 8);

for(int j = 0; j <= frame.cols; j += frame.cols/5)

line(frame, cv::Point(j, 0), cv::Point(j, frame.rows), cv::Scalar(0, 255, 0), 2, 8);

remap(frame, undist, map1, map2, cv::INTER\_LINEAR);

QPixmap saveImg = QPixmap::fromImage(

QImage(undist.data,

undist.cols,

undist.rows,

undist.step,

QImage::Format\_RGB888).rgbSwapped());

QString undistDir;

if(numCam == 1)

undistDir = fs\_->getFilePath() + "Camera1/" + "Undistorted/" + QString::number(i+1) + ".png";

else if(numCam == 2)

undistDir = fs\_->getFilePath() + "Camera2/" + "Undistorted/" + QString::number(i+1) + ".png";

QFile file(undistDir);

file.open(QIODevice::WriteOnly);

saveImg.save(&file, "png");

imageInfo[i].undistortedPath = undistDir.toStdString();

file.close();

}

if(numCam == 1)

fs\_->saveInfoCamera1(imageInfo);

else if(numCam == 2)

fs\_->saveInfoCamera2(imageInfo);

}

void CalibrationProcessor::setTargetType(QString qstring)

{

targetType\_ = qstring;

}