

Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

**ОТЧЁТ О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ
РАБОТЕ**

Система стереозрения на основе ортогонально ориентированных камер

Выполнил
студент гр.3331506/80401 <подпись> М. Д. Пантелейев

Руководитель
старший преподаватель <подпись> А. С. Габриель

Научный консультант <подпись> В. В. Варлашин

«___» _____ 202__ г.

Санкт-Петербург
2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1 Обзор опыта построения систем стереозрения с использованием сверхширокоугольных камер	7
1.1 Модель сверхширокоугольной камеры	7
1.2 Обзор существующих систем стереозрения, использующих сверхширокоугольные камеры	10
1.3 Обоснование выбора ПО	11
1.4 Выводы по главе	13
2 Система стереозрения	14
2.1 Алгоритм устранения искажений	14
2.2 Описание системы стереозрения	17
2.3 Виртуальное моделирование системы	19
2.4 Передача изображений с виртуальных камер для обработки . .	21
2.5 Выводы по главе	23
3 Экспериментальное исследование системы стереозрения	24
3.1 Оценка качества устранения искажений	24
3.2 Оценка отклонения облака точек от поверхности	26
3.3 Выводы по главе	28
ЗАКЛЮЧЕНИЕ	29
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	30
ПРИЛОЖЕНИЕ А	32
ПРИЛОЖЕНИЕ Б	35
ПРИЛОЖЕНИЕ В	46
ПРИЛОЖЕНИЕ Г	56

ВВЕДЕНИЕ

За последние годы был совершен существенный прогресс в доступности и точности сенсоров, позволяющих мобильным роботам осуществлять оценку окружающего пространства. Такие информационно-измерительные устройства как лидары, сонары и стереокамеры стали основной опорой алгоритмов для алгоритмов автономной навигации и локализации. Тем не менее в роботах по-прежнему присутствуют оптические системы, так как они дают наиболее читаемую информацию для оператора в случаях, когда его вмешательство необходимо. Многие современные мобильные роботы имеют у себя на борту камеры с сверхшироким ($> 90^\circ$) углом зрения, так как они, хоть и вносят искажения в воспринимаемую картину, позволяют охватить больше окружающего пространства. Набор таких камер может составлять систему кругового обзора [], позволяющую оператору видеть не только в любом направлении, но даже с видом от третьего лица [7].

В случае автономных мобильных роботов подобные системы включаются лишь по необходимости, но при этом могут быть весьма дорогостоящими и занимать место в корпусе. Широкоугольные камеры в системах кругового обзора роботов часто имеют области пересечения полей зрения, что позволяет проводить оценку глубины. Реализация системы стереозрения, способной работать в таких конфигурациях, позволит дать существующим роботам новый способ получать информацию об окружении и проектировать будущих роботов с учётом этой возможности.

Однако значительные радиальные искажения изображения, вызванные особенностями используемых объективов, вместе с тем фактом, что области пересечения обычно расположены ближе к краям изображения, не позволяют использовать известные алгоритмы стереозрения.

Целью работы является разработка и изучение точности системы стереозрения, основанной на ортогонально расположенных сверхширокоугольных камерах.

В ходе работы решаются следующие задачи:

- Обзор современных систем стереозрения, использующих изображения с широкоугольных камер.
- Обзор моделей, применяющихся в моделировании камер с объективами ”рыбий глаз”.
- Обоснование выбора программного обеспечения, используемого для разработки и тестирования алгоритма.
- Разработка алгоритма устранения искажений fisheye-изображения в любой области кадра.
- Моделирование системы стереозрения в виртуальной среде.
- Определение точности оценки глубины стереосистемой.

1 Обзор опыта построения систем стереозрения с использованием сверхширокоугольных камер

1.1 Модель сверхширокоугольной камеры

Сверхширокоугольные объективы имеют в своей основе сложную систему линз, схема которой вместе с примером изображения представлена на рисунке 1.1. Особенности этой системы позволяют достигать существенного угла обзора, но также являются причиной aberrации и характерных искажений изображения. Моделировать реальный ход лучей в подобных камерах нецелесообразно, поэтому исследователи прибегают к моделям камер.

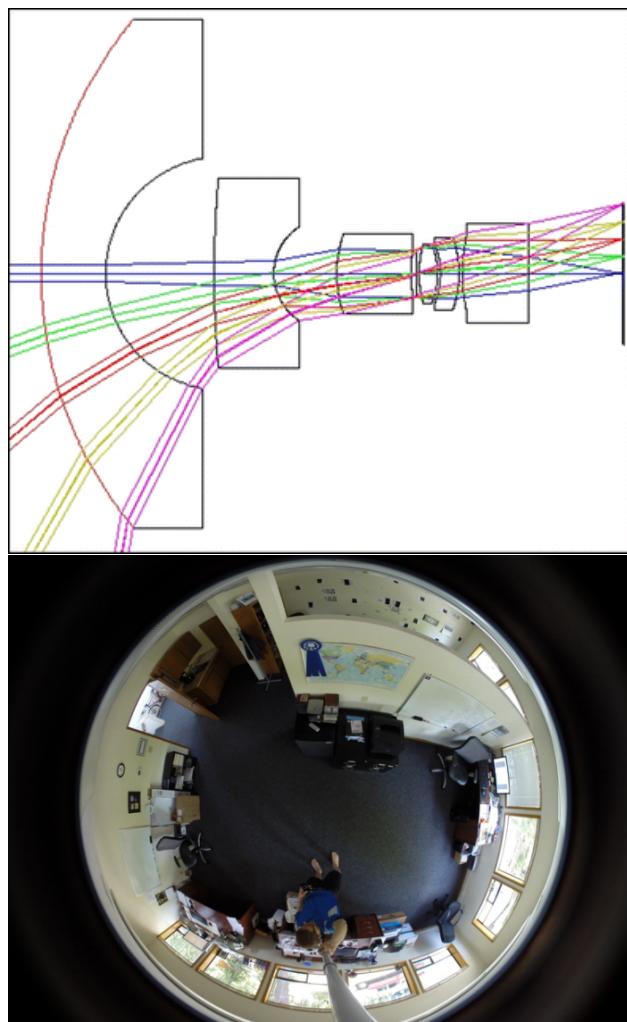


Рисунок 1.1 – Схема хода лучей объектива ”рыбий глаз” (слева), пример изображения (справа)??

Как видно по рисунку 1.2, модель проекции для камеры это функция (обычно обозначаемая $\pi_c()$), которая моделирует преобразование из точки

трёхмерного пространства ($P = [x_c, y_c, z_c]^T$) в области зрения камеры в точку на плоскости изображения ($p = [u, v]^T$). Единичная полусфера S с центром в точке O_c описывает поле зрения. На ней также лежит точка P_C , являющаяся результатом обратной проекции $\pi_c^{-1}(p)$. Угол θ является углом падения для рассматриваемой точки, а угол ϕ откладывается между положительным направлением оси x и O_ip .

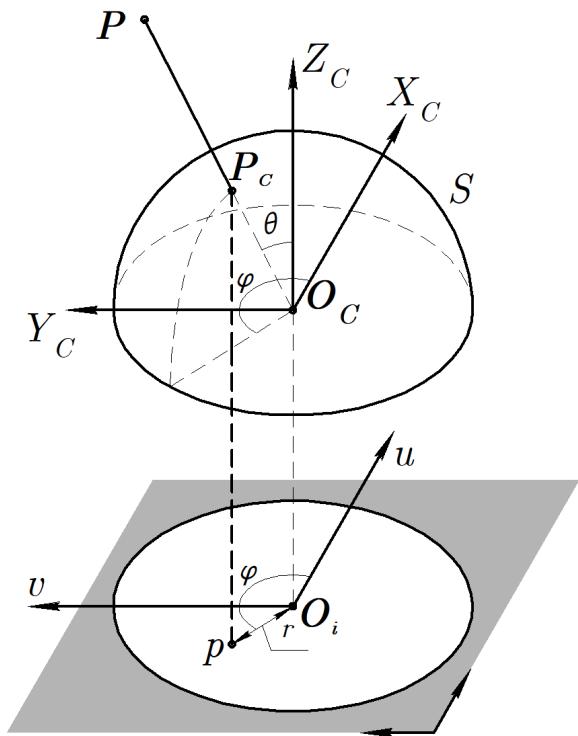


Рисунок 1.2 – Схема проекции точки трёхмерного пространства в точку на изображении

Модели камер включают в себя описания нескольких типов искажений, но в сверхширокоугольных объективах самыми существенными являются радиальные - искажения, проявляющиеся сильнее ближе к краям изображения. Поэтому далее в этой секции модели будут рассматриваться именно с точки зрения описания радиальных искажений.

Перспективная проекция, которая обычно используется в качестве модели ортоскопической камеры, не способна спроектировать широкоугольное пространство на кадр конечного размера. Поэтому при описании и проектировании fisheye-объективов опираются на другие виды проекций [6]. Но

реальные линзы не всегда в точности следуют заданным моделям, к тому же отличия в используемых параметрах усложняют процесс калибровки камер. По этой причине радиальные искажения принято аппроксимировать многочленами, например, вида

$$\delta r = k_1 r^3 + k_2 r^5 + k_3 r^7 + \dots + k_n r^{n+2}, \quad (1.1)$$

где k_i - коэффициенты, описывающие внутренние параметры камеры.

В настоящий момент есть несколько распространённых моделей, аппроксимирующих реальные искажения подобных объективов. Модель Канналы и Брандта [5] реализована в OpenCV и описывает радиальные искажения через угол падения луча света на линзу, а не расстояние от центра изображения до места падения, как это делалось в более ранних моделях. Авторы посчитали, что для описания типичных искажений достаточно пяти членов полинома. Таким образом, указанную модель можно записать следующими уравнениями:

$$\theta = \arctan\left(\frac{r}{f}\right), \quad (1.2)$$

$$\delta r = k_1 \theta + k_2 \theta^3 + k_3 \theta^5 + k_4 \theta^7 + k_5 \theta^9, \quad (1.3)$$

$$\begin{pmatrix} u \\ v \end{pmatrix} = \delta r(\theta) \begin{pmatrix} \cos(\phi) \\ \sin(\phi) \end{pmatrix}, \quad (1.4)$$

где θ - угол падения луча, определяемый выбранным типом проекции; ϕ - угол между горизонтом и проекцией падающего луча на плоскость изображения; r - расстояние от спроектированной точки до центра изображения; f - фокусное расстояние.

Также большое распространение получила модель Скамуззы [9], которая легла в основу Matlab Omnidirectional Camera Calibration Toolbox. Она связывает точки на изображении с соответствующей им точкой в координатах

камеры следующим образом

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = \lambda \begin{pmatrix} u \\ v \\ a_0 + a_2 r^2 + a_3 r^3 + a_4 r^4 \end{pmatrix}, \quad (1.5)$$

где $a_0 \dots a_4$ - коэффициенты, описывающие внутренние параметры камеры; λ - масштабный коэффициент.

1.2 Обзор существующих систем стереозрения, использующих сверхширокоугольные камеры

Распространённые библиотеки машинного зрения (OpenCV, MATLAB CV Toolbox) предлагают готовые к использованию классы и функции, позволяющие после калибровки камер получать с помощью них карты глубины. Однако на практике эти методы весьма ограничены. Для стереосопоставления используются традиционные методы, приспособленные для классических камер с перспективной проекцией, что не позволяет использовать кадры с широкоугольных камер целиком. В результате у этих кадров после устранения искажений остаётся угол зрения порядка 90° . Кроме того, библиотечные функции не позволяют задать область интереса для каждой камеры, что ограничивает их область применения только для копланарного расположения камер.

Исследователи предложили несколько реализаций стереозрения, опирающихся на снимки со сверхширокоугольных объективов и лишённых недостатков распространённых решений. Например, метод, предложенный в [2], позволяет создать кольцевую область стереозрения с вертикальным полем зрения 65° . Для этого используются две 245° камеры, закреплённые на противоположных концах жёсткого стержня. Это позволяет достигнуть панорамного обзора глубины с качеством, достаточным для осуществления автономной навигации и локализации [1], но доступна такая схема расположения камер только летательным аппаратам.

Другие авторы [8] решили отказаться от типичных для стереозрения этапов устранения искажений и ректификации и извлекать информацию о глубине напрямую по двум снимкам fisheye-камер. Для производства карт глубины используется свёрточная нейронная сеть, что требует существенных вычислительных мощностей - для достижения производительности в реальном времени разработчикам понадобилось использовать компьютер с ЦПУ i7-4770 и ГПУ NVIDIA GTX 1080Ti. В мобильном автономном роботе такой вычислитель разместить может быть проблематично. Кроме того, метод так же предполагает, что обе камеры направлены в одном направлении.

1.3 Обоснование выбора ПО

Разработку и первоначальные испытания алгоритма стереозрения целесообразно проводить в виртуальной среде. Это позволяет значительно упростить разработку, так как уменьшает время на проверку гипотез и расходы на реальное оборудование, особенно в случае неудачных испытаний. Из-за этих факторов виртуальное моделирование в робототехнике приобрело широкое распространение и активно применяется, например, для разработки систем локализации и навигации беспилотного транспорта []. Возросшее качество компьютерной графики к тому же позволило моделировать реалистичное окружение, что особенно важно при работе с системами технического зрения.

Для разработки алгоритма, описанного в этой работе, нужна виртуальная среда, в которой можно симулировать несколько широкоугольных камер и настраивать их параметры, легко интегрировать алгоритмы технического зрения и создать окружение, приближенное к тому, в котором будет работать алгоритм. На данный момент исследователю доступен широкий выбор программного обеспечения, подходящего для этой задачи. В таблице 1.1 представлено сравнение имеющихся предложений по основным изложенным выше требованиям.

Таблица 1.1 – Сравнение ПО для симуляции

Название	Симуляция fisheye-камер	Реалистичное моделирование	Интеграция кода	Доступ
Gazebo	Возможна	Затруднено	Возможна посредством ROS	Бесплат
RoboDK	Нет	Затруднено	Нет	От 145€
Webots	Затруднена	Возможно	Возможна	Бесплат
CoppeliaSim	Затруднена	Затруднено	Возможна	Бесплат
NVIDIA Isaac Sim	Возможна	Возможно	Возможна	Бесплат
CARLA	Затруднена	Возможно	Возможна	Бесплат
Unity	Возможна	Возможно	Возможна	Бесплат

По результатам оценки собранные сведения принято решение проводить разработку в симуляторе Unity. Он позволяет подробно настраивать камеру и эмулировать fisheye-объектив, строить реалистичные сцены благодаря свободному импорту моделей, а при программировании в симуляторе можно использовать сторонние программы в виде динамически подключаемых библиотек. По функционалу так же подходит NVIDIA Isaac Sim, но от него пришлось отказаться из-за высоких системных требований и малой изученности продукта.

Разрабатываемое решение должно иметь возможность внедрения в ПО робота, поэтому должно реализовываться на одном из популярных и быстродейственных языков программирования. Учитывая необходимость интеграции с Unity и потребность использовать популярные библиотеки, выбран язык C++. Другим важным фактором является библиотека обработки изображений. В качестве основы для программной части была выбрана библиотека OpenCV, являющаяся стандартом при разработке систем технического зрения. Она доступна к использованию со множеством языков

программирования, но наилучшую производительность показывает именно с C++ [].

1.4 Выводы по главе

Обзор современных моделей сверхширокоугольных камер позволил выбрать наиболее точную и удобную для калибровки. Был осуществлён обзор существующих систем стереозрения, применяющих камеры типа "рыбий глаз" с целью ознакомления с мировым опытом. Принято решение разрабатывать и тестировать предлагаемую систему стереозрения с применением виртуального моделирования.

Для моделирования выбрана среда разработки Unity. Для обработки изображений с камер выбрана библиотека OpenCV для языка программирования C++.

2 Система стереозрения

2.1 Алгоритм устранения искажений

Как уже упоминалось в секции 1.1, существующие способы устранения радиальных искажений не позволяют работать близко к краям изображений, поэтому для реализации предлагаемой системы стереозрения был разработан алгоритм устранения искажений на основе модели [9]. Схема геометрического принципа, лежащего в основе этого алгоритма, представлена на рисунке 2.1.

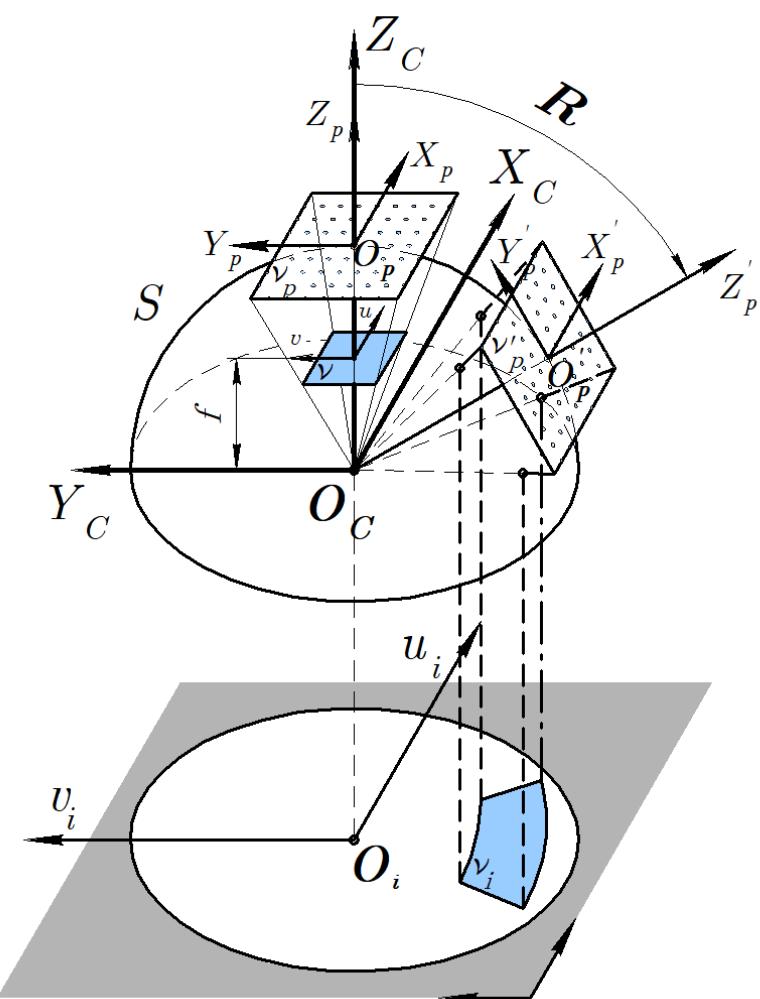


Рисунок 2.1 – Схема принципа устранения искажений

Цель алгоритма - найти, куда на выходном изображении проектируются все пиксели из выбранного участка входного сверхширокоугольного изображения. Сделать это можно, выполнив обратное преобразование 1.5 для каждого пикселя fisheye-снимка и затем прямое преобразование модели

камеры-обскуры для получения результирующей проекции. Однако такой подход приведёт к возникновению дефектов из-за несовпадения частот дискретизации двух изображений. Поэтому вместо этого алгоритм выполняет обратное преобразование для каждого пикселя итогового изображения ν , находя таким образом соответствующую ему точку в системе координат камеры (X_c, Y_c, Z_c)

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} u * z_c / f \\ v * z_c / f \\ z_c \end{bmatrix}, \quad (2.1)$$

где f - фокусное расстояние.

Набор таких точек формирует прямоугольную область ν_p с центром в точке O_p и является виртуальной камерой-обскурой с оптической осью Z_p . Поворот точек, входящих в ν_p , с помощью матрицы вращения R образует ν'_p и позволяет таким образом задать направление обзора и ориентацию виртуальной камеры.

$$\begin{bmatrix} x'_p \\ y'_p \\ z'_p \end{bmatrix} = \begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} R. \quad (2.2)$$

$$R = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \beta & \sin \beta \\ 0 & -\sin \beta & \cos \beta \end{bmatrix} \begin{bmatrix} \cos \gamma & 0 & -\sin \gamma \\ 0 & 1 & 0 \\ \sin \gamma & 0 & \cos \gamma \end{bmatrix}, \quad (2.3)$$

где α, β, γ - углы Эйлера.

Тогда обратная fisheye-проекция точек из ν'_p позволяют получить область ν_i исходного изображения с искомыми пикселями. Таким образом, зная как геометрически проектируется каждая точка из ν в ν_i , можно перенести информацию о цвете и получить изображение с устранимыми радиальными искажениями в любой части поля зрения камеры.

Так как в используемой модели искажения в пикселе считаются центрально симметричными и зависят только от её удаления от центра изображения, рассмотрим ход падающего луча в координатах (Z_c, ρ) , изображённый на рисунке 2.2.

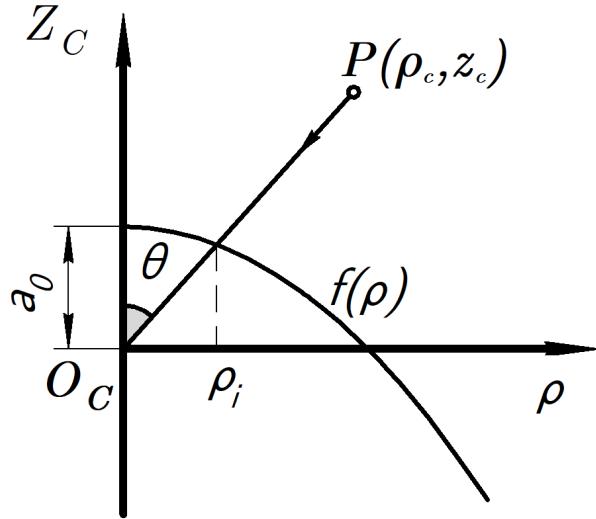


Рисунок 2.2 – Нахождение обратной проекции в используемой проекции

Для модели (1.5) обратная проекция записывается как

$$\begin{bmatrix} u_i \\ v_i \end{bmatrix} = \begin{bmatrix} \frac{x_c}{\lambda} \\ \frac{y_c}{\lambda} \end{bmatrix}, \quad (2.4)$$

где $\lambda = \rho_c / \rho_i$ - масштабный коэффициент.

Для нахождения ρ_i был применён метод последовательных приближений. Блок-схема алгоритма приведена на рисунке 2.3.

SAMPLE

Рисунок 2.3 – Блок-схема алгоритма поиска ρ

Очевидно, весь процесс преобразования $v \rightarrow v_i$ требует выполнения существенного количества математических операций, что негативно сказывается на скорости, с которой алгоритм может обрабатывать изображения в реальном времени. Однако при неизменных параметрах модели алгоритм достаточно выполнить лишь один раз, записав результат в таблицу поиска - структуру данных, которая позволяет дальнейшие преобразования проводить по уже известным соотношениям между пикселями. Время построения таблицы поиска для выходного изображения разрешением 540*540 пикселей

и входного изображения 1080×1080 пикселей составляет $\sim 2.2\text{с}$. Это время можно уменьшить, применив различные оптимизации и параллельные вычисления (проекция каждого пикселя может рассчитываться независимо). Время отображения с использованием таблиц поиска составляет для того же изображения ..., что позволяет применять алгоритм для устранения искажений в реальном времени.

2.2 Описание системы стереозрения

Предлагаемая система стереозрения состоит из двух или более камер с объективами типа "рыбий глаз" $\geq 180^\circ$, расположенных ортогонально. В таком случае поля зрения соседних камер могут пересекаться, образуя области, точки пространства в которых видны на обеих камерах. Это позволяет получить информацию о глубине в указанных областях.

Робот-доставщик "Ровер" компании Яндекс имеет на борту 4 сверхширокоугольные камеры, размещённых спереди, сзади и по бортам корпуса робота, что позволяет рассмотреть принцип работы системы стереозрения на его примере. На рисунке 2.4 представлено сравнение систем стереозрения. Слева изображена схема зон (обозначены зелёным), в которых обеспечивается получение информации о глубине при использовании описываемой системы, а справа эквивалентная по горизонтальному покрытию схема при использовании обычных камер с углом обзора 90° .

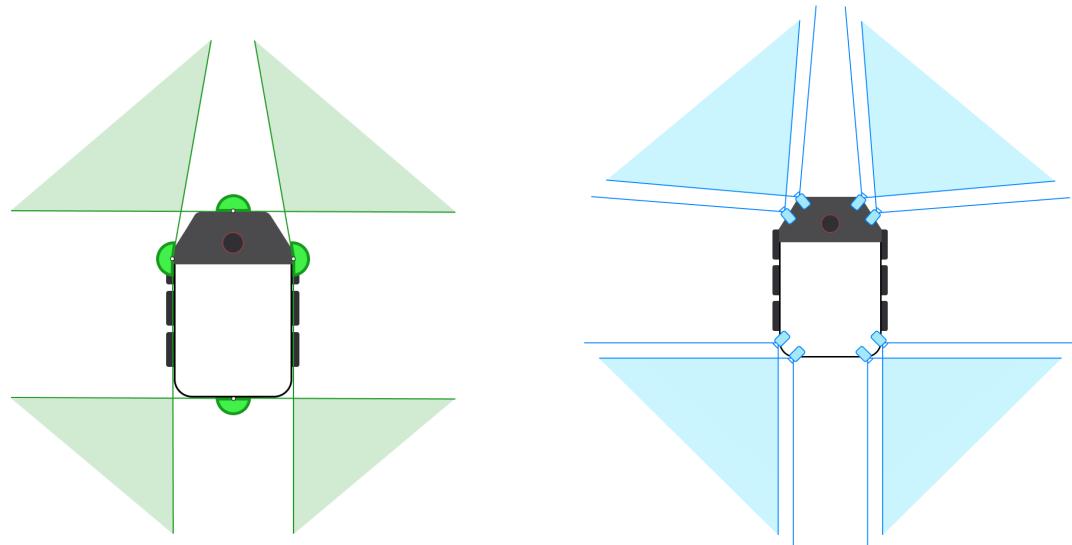


Рисунок 2.4 – Сравнение систем стереозрения

Как можно заметить, системе на основе обычных камер нужно в два раза больше сенсоров, чтобы достичь той же зоны покрытия по горизонтали. Кроме того, традиционная система имеет меньшую зону покрытия по вертикали и не обеспечивает полный панорамный обзор. Все эти факторы делают систему стереозрения на основе ортогонально расположенных сверхширокоугольных камер более предпочтительной для применения в робототехнике.

Применение традиционных алгоритмов стереозрения предполагает наличие стереопары - двух камер с известным взаимным положением, наблюдающих одну область пространства с разных ракурсов. Описанный в предыдущей секции метод устранения искажений сверхширокоугольной камеры позволяет создавать виртуальные камеры-обскуры и регулировать их направление обзора. Таким образом, стереопару можно сформировать из двух таких виртуальных камер. Далее для упрощения рассмотрения системы будет считаться, что оптические оси всех камер находятся в одной плоскости, а на ориентацию виртуальных камер влияет только один угол.

На рисунке 2.5 изображён вариант системы с двумя камерами под углом 90° , соответствующий фрагменту схемы, представленной на рисунке 2.4.

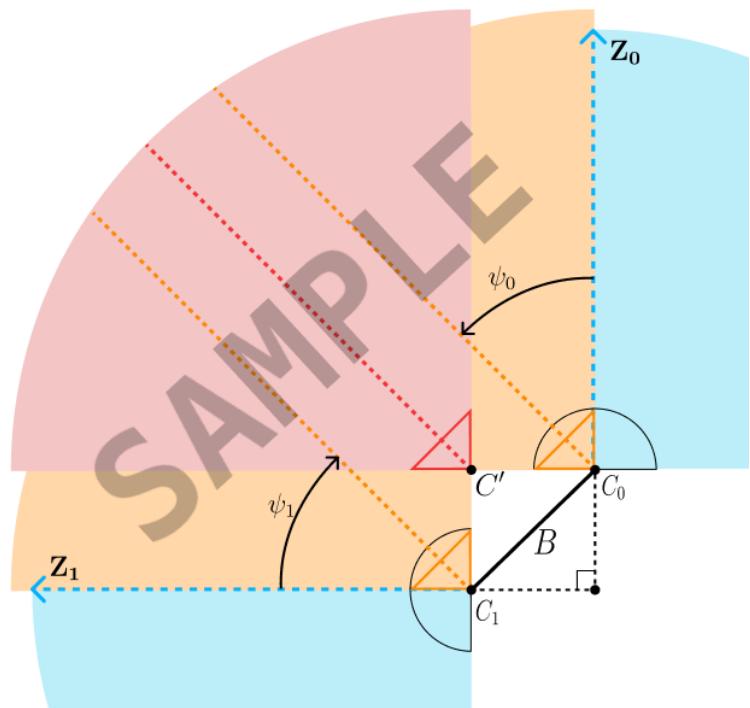


Рисунок 2.5 – Геометрическая модель бинокулярной системы стереозрения

Здесь у полей зрения камер C_0 и C_1 есть область пересечения, обозначенная красным. Эта область эквивалентна области пересечения полей зрения двух камер с полями зрения 90° (обозначены оранжевым), повернутых на 45° в сторону области интереса. В таком случае B - база стереопары. Примеры изображений, полученных в такой конфигурации, изображены на рисунке 2.6.

SAMPLE

Рисунок 2.6 – Пример исходных изображений и снимков виртуальной стереопары

2.3 Виртуальное моделирование системы

Описанная система была смоделирована в среде Unity, её внешний вид представлен на рисунке 2.7. Мир Unity предназначен для базовой проверки работоспособности испытываемого принципа, поэтому не содержит подробной модели какого-либо робота. В нём присутствуют: плоскость земли, кронштейн, на котором сверхширокоугольные камеры закреплены под углом 90° , подвижный калибровочный узор и объекты-цели, предназначенные для оценки расстояния. В силу особенностей работы многих алгоритмов стереоразрезания эти объекты должны сильнотекстурированы [].



Рисунок 2.7 – Внешний вид сцены в Unity

В Unity создание сцены происходит с использованием встроенных примитивов, импортированных файлов моделей популярных форматов или моделей из магазина ассетов, предлагающего обширную библиотеку объектов и текстур, повторяющих различные реальные объекты. В данной работе использовались как и примитивы для создания простых объектов, так и модели из магазина для имитации препятствий и окружения.

Для моделирования камеры "рыбий глаз" использовался аддон Dome Tools из магазина Unity Asset Store. Он позволяет моделировать сверхширокоугольные объективы с разным углом зрения в эквидистантной проекции []. При этом с точки зрения других искажений, не относящихся к моделированию правильной проекции, снимки с этой камеры получаются идеальными. Окно настроек камеры представлено на рисунке 2.8.

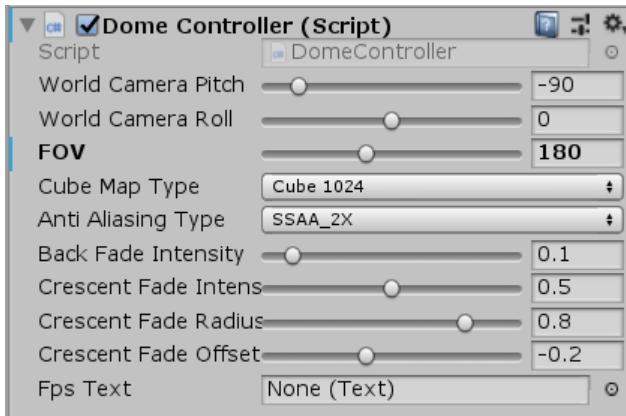


Рисунок 2.8 – Окно настроек виртуальной камеры в Unity

Для виртуальной камеры доступны настройки угла зрения и виньетки по краям изображения, а также различных параметров рендеринга, влияющих на качество получаемого изображения.

2.4 Передача изображений с виртуальных камер для обработки

Unity в качестве основного языка программирования использует C# и не позволяет импортировать выбранную библиотеку технического зрения OpenCV напрямую. Однако как уже упоминалось в секции 1.3, данная среда позволяет интегрировать плагины, в том числе написанные на других языках программирования, в виде динамически подключаемых библиотек (DLL). Именно этот подход был использован для передачи изображений - была разработана библиотека с функциями для обработки снимков, которая компилировалась отдельно и затем импортировалась в Unity.

Для реализации описанного в этой работе принципа и упрощения разработки в библиотеке были реализованы следующие функции:

- initialize - выделяет память под нужные структуры, заполняет таблицу поиска и создаёт окна для отображения будущих изображений.
- getImages - передаёт изображения в библиотеку и производит обработку изображений в соответствии с аргументами.
- takeScreenshot - производит ту же обработку входного изображения, что и предыдущая функция, но результат сохраняет в файл.
- processImage - передаёт обратно в скрипт уже обработанные изображения для удобного отображения в интерфейсе Unity.

- `terminate` - высвобождает память по завершении работы симуляции.

Обработка, упомянутая в пунктах с функциями `getImages` и `takeScreenshot`, заключается в конвертации цветового пространства изображений (из `RGBA`, используемого в Unity, в `BGR`, используемый в OpenCV), горизонтальном зеркальном отображении (из-за разного положения точки отсчёта координат пикселей) и, наконец, устранив искажений в области интереса. Так как для наилучшей работы стереосопоставления снимки должны быть синхронизированы (то есть получены камерами в один момент времени), обработка производится сразу для двух изображений. Программный код файлов, входящих в динамически подключаемую библиотеку, представлен в приложении ??.

Также написан специальный скрипт, который управляет процессом взаимодействия симуляции и библиотеки, его код приведён в приложении ?. Эта программа имеет окно настроек, которое позволяет задать используемые камеры и элементы управления параметрами, выбрать область интереса и ... Внешний вид этого окна представлен на рисунке 2.9.

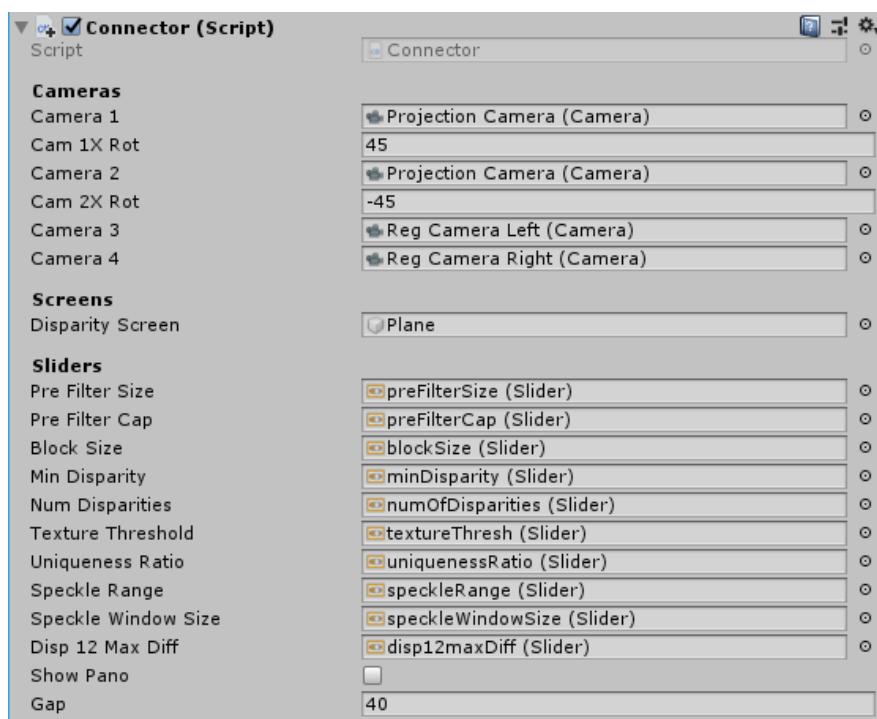


Рисунок 2.9 – Внешний вид окна настроек скрипта

Сразу после запуска симуляции происходит запуск дополнительных вычислительных потоков для обработки кадров с каждой пары камер и

передача настроек скриптом в библиотеку для построения таблиц поиска. Использование нескольких потоков позволяет не блокировать работу симуляции на время обработки изображений. Далее каждое обновление кадра скрипт считывает изображения с камер и помещает их в память соответствующего потока. Поток же работает независимо и обрабатывает каждую следующую пару изображений после готовности предыдущей.

2.5 Выводы по главе

Описан принцип устранения искажений сверхширокоугольных линз с выбором области интереса. Разработан алгоритм нахождения обратной проекции для fisheye-изображения. Описано устройство системы стереовидения. Разработана её виртуальная модель в среде Unity вместе с алгоритмом передачи изображений с виртуальной камеры в программу обработки изображений... Виртуальная модель системы позволяет перейти к её испытаниям.

3 Экспериментальное исследование системы стереозрения

Описанная в предыдущей главе виртуальная модель системы стереозрения позволяет проводить с ней испытания для оценки работоспособности и сравнения с аналогами в контролируемой среде.

Результатом работы алгоритма стереозрения является карта глубины воспринимаемого пространства. Подобные карты могут использоваться, например, алгоритмами навигации и локализации для построения карты окружения робота. Точность работы этих алгоритмов зависит от того, насколько точно карта глубины передаёт реальную информацию о форме объектов. Оценить эту характеристику для отдельной системы стереозрения проблематично, так как она зависит от множества факторов. Поэтому оценка качества работы системы проведена в сравнении с виртуальной моделью традиционной стереопары.

Виртуальная сцена позволяет разместить сразу несколько объектов в одной точке пространства, таким образом возможно в существующую модель в Unity добавить ещё 2 камеры, совпадающие по параметрам и положению с виртуальными камерами на рисунке 2.5. Разрешение этих камер выбрано исходя из размеров проекции ν_i области интереса на широкоугольном снимке. Для камер "рыбий глаз" с разрешением 1080 * 1080 пикселей она составляет 287482 пикселяй, что аналогично камере с разрешением 540 * 540. В результате получена эталонная стереопара для сравнения.

3.1 Оценка качества устранения искажений

Добавление в виртуальную модель эталонной камеры позволяет оценить качество устранения искажений путём сравнения её снимков со снимками виртуальной камеры-обскуры модуля устранения искажений.

Параметры полинома (1.5) для устранения искажений получены с помощью MATLAB Camera Calibrator и занесены в код библиотеки обработки изображений. Снимки после устранения искажений и с эталонной камеры показаны на рисунке 3.1.



Рисунок 3.1 – Слева - снимок после устранения искажений; справа - эталонный снимок.

На левом изображении заметна большая резкость, вызванная, вероятно, округлениями чисел в процессе проекции. Также присутствует затенение по левому краю - следствие аберрации на исходном снимке. В той же области есть малозаметные искажения геометрии. Более явно эти дефекты можно увидеть на разностном изображении, представленном на рисунке 3.2.



Рисунок 3.2 – Разностное изображение

Анализ этого изображения подтверждает различия в резкости и наличие искажений в левой части снимка. Тем не менее, искажения достаточно несущественны и проявляются лишь близь краёв исходного изображения,

что позволяет считать подобные снимки пригодными к применению в системе стереозрения. Кроме того, некоторые эффекты возможно устраниить или сильно ослабить более качественной калибровкой камер...

3.2 Оценка отклонения облака точек от поверхности

В качестве целевой поверхности выбрана виртуальная плоскость с нанесённой на неё текстурой высокого разрешения. Размеры и положение плоскости относительно камер известно с высокой точностью, что позволяет сравнить результаты стереореконструкции с реальным положением целевого объекта и оценить ошибку. В качестве метрики оценки выбрано среднее квадратичное отклонение положения точек от модели плоскости.

Исследования проведены в MATLAB, снимки получены с помощью виртуальной модели. Сначала для обеих стереопар проводится калибровка по снимкам узора шахматной доски с помощью методики [3], что позволяет получить внутренние и внешние параметры камер. Далее происходит построение карты расхождений [4] и 3D-реконструкция сцены. Результатирующее облако точек представлено на рисунке 3.3.

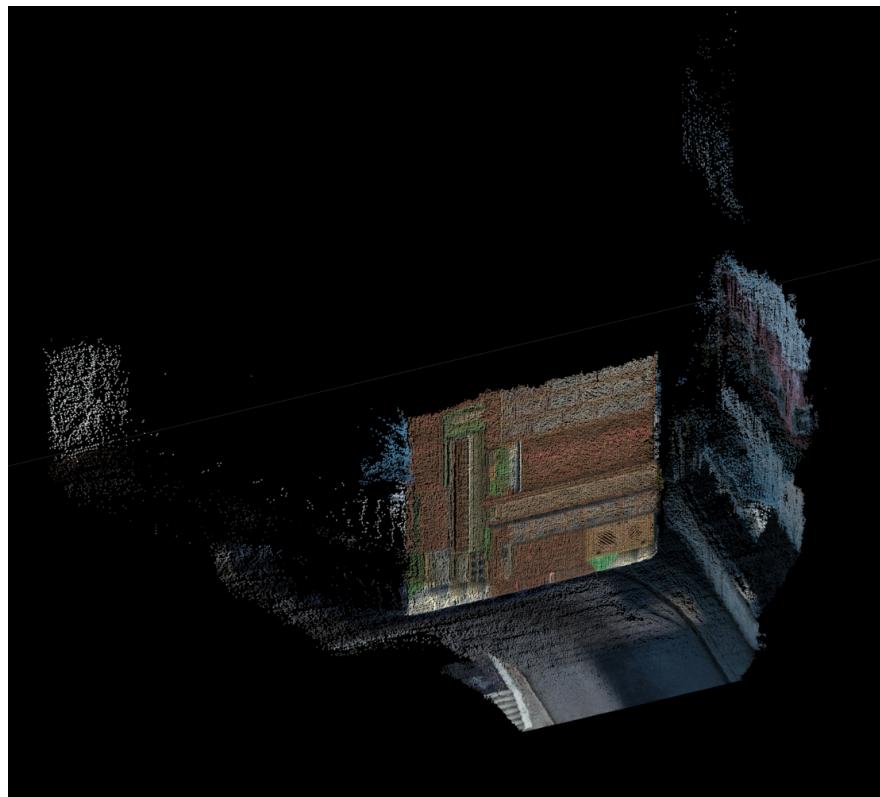


Рисунок 3.3 – Неочищенное облако точек

По заданным параметрам строится модель целевой плоскости, а точки за пределами её окрестности отбрасываются. Затем скрипт перебирает все оставшиеся точки и рассчитывает среднее квадратичное отклонение по длине нормали от точки к плоскости. Код скрипта приведён в приложении Д.

Описанный алгоритм применён к снимкам целевой плоскости на разных расстояниях с эталонной и исследуемой стереопар. Результаты приведены на графике 3.4.

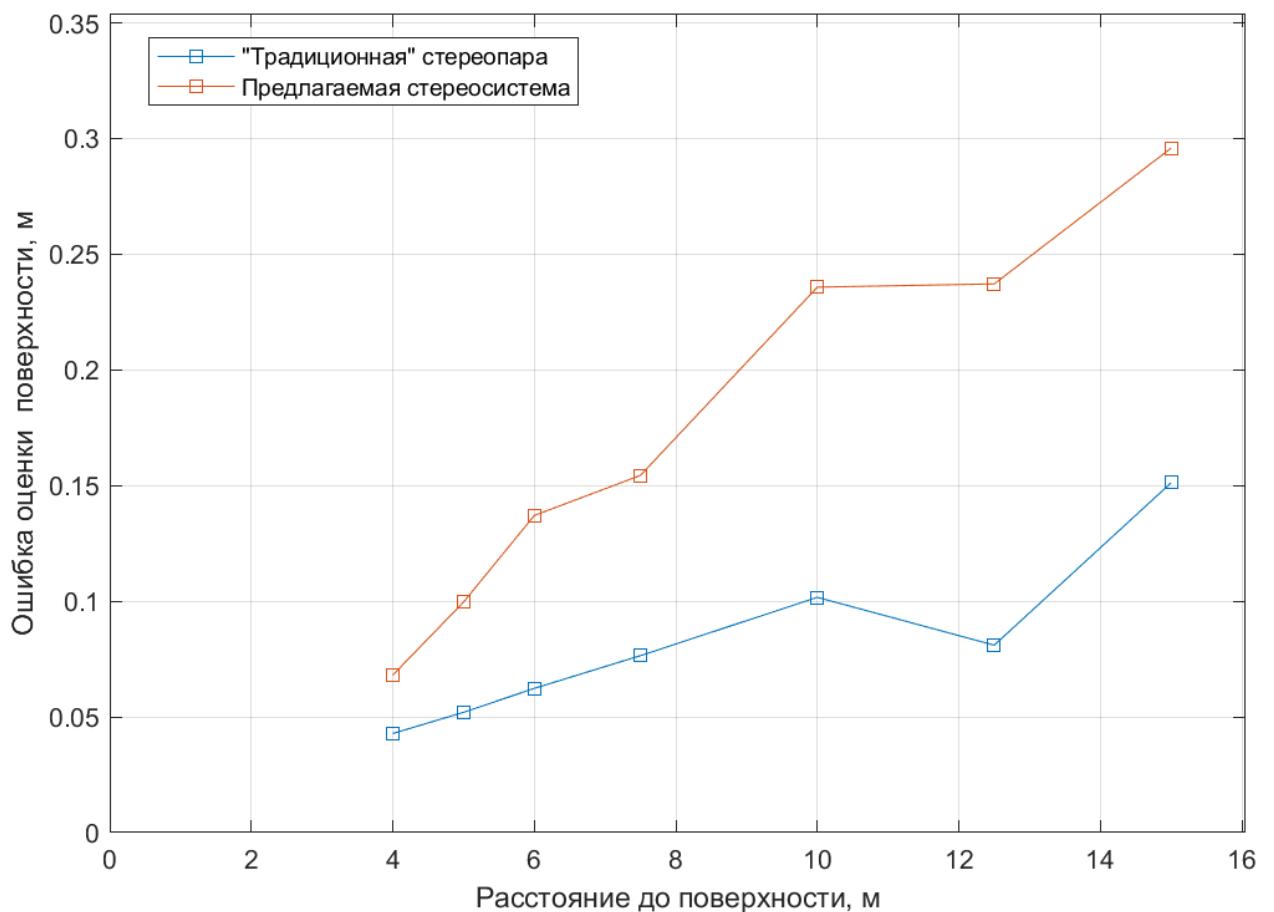


Рисунок 3.4 – График среднеквадратичной ошибки в зависимости от расстояния для эталонной и исследуемой стереопар

Как видно из графика, ошибка оценки поверхности в исследуемой системе в среднем в два раза больше этого показателя для традиционной стереопары. Меньшая точность исследуемой системы связана с влиянием факторов, описанных в предыдущей секции, на алгоритм стереосопоставления. Тем не менее она остаётся в пределах 3% от реального расстояния.

3.3 Выводы по главе

Исследованы снимки, полученные с помощью виртуальных камер. Выполнено их сравнение с эталонными изображениями, которое продемонстрировало пригодность снимков для использования в системах стереозрения.

В виртуальную среду добавлена традиционная стереопара для сравнения с разработанной системой. Проведён эксперимент по оценке качества облака точек, полученного с помощью предлагаемого решения. Результаты можно считать удовлетворительными для систем подобного класса.

ЗАКЛЮЧЕНИЕ

В ходе работы рассмотрены существующие системы стереозрения, в особенности использующие информацию со сверхширокоугольных камер. Изучены модели, описывающие подобные камеры. Рассмотрено ПО для виртуального моделирования робототехнических комплексов и систем технического зрения. Выбрано ПО для моделирования и разработки системы стереозрения, использующей объективы "рыбий глаз".

Разработан и описан алгоритм устранения искажений в области интереса и его математическая модель. Описана возможная конфигурация системы стереозрения, рассмотрен принцип работы её фрагмента. Этот фрагмент считать был смоделирован в среде Unity. В процессе размещены и настроены камеры, подготовлены объекты для калибровки и дальнейших испытаний системы. Реализована передача изображений с виртуальных камер для обработки алгоритмами компьютерного зрения. С помощью снимков из виртуальной модели исследовано качество устранения искажений и выполнено сравнение точности оценки глубины предлагаемой системы и варианта с обычными камерами. Результаты позволяют рассмотреть применение системы с реальными камерами и указывают на возможные доработки.

Дальнейшая работа будет сконцентрирована на оптимизации и повышении точности алгоритма устранения искажений, разработке способа автоматической установки параметров системы при разных конфигурациях камер. После этого можно будет приступить к испытаниям качества построения карты и локализации с применением описанной системы сначала с виртуальным, а затем и с реальным роботом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Autonomous aerial robot using dual-fisheye cameras / W. Gao [и др.] // Journal of Field Robotics. — 2020. — Т. 37, № 4. — С. 497—514.
2. Gao W., Shen S. Dual-fisheye omnidirectional stereo // 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). — 2017. — С. 6715—6722. — DOI: 10.1109/IROS.2017.8206587.
3. Heikkila J., Silven O. A four-step camera calibration procedure with implicit image correction // Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition. — 1997. — С. 1106—1112. — DOI: 10.1109/CVPR.1997.609468.
4. Hirschmuller H. Accurate and efficient stereo processing by semi-global matching and mutual information // 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05). Т. 2. — 2005. — 807—814 vol. 2. — DOI: 10.1109/CVPR.2005.56.
5. Kannala J., Brandt S. A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses // IEEE Transactions on Pattern Analysis and Machine Intelligence. — 2006. — Т. 28, № 8. — С. 1335—1340. — DOI: 10.1109/TPAMI.2006.153.
6. Miyamoto K. Fish Eye Lens // J. Opt. Soc. Am. — 1964. — Авг. — Т. 54, № 8. — С. 1060—1061. — DOI: 10.1364/JOSA.54.001060. — URL: <http://www.osapublishing.org/abstract.cfm?URI=josa-54-8-1060>.
7. Real-Time Bird's Eye Surround View System: An Embedded Perspective / M. Al-Hami [и др.] // Applied Artificial Intelligence. — 2021. — Т. 35, № 10. — С. 765—781. — DOI: 10.1080/08839514.2021.1935587.
8. Roxas M., Oishi T. Real-Time Variational Fisheye Stereo without Rectification and Undistortion. — 2019. — arXiv: 1909.07545 [cs.RO].

9. *Scaramuzza D., Martinelli A., Siegwart R.* A Flexible Technique for Accurate Omnidirectional Camera Calibration and Structure from Motion // Fourth IEEE International Conference on Computer Vision Systems (ICVS'06). — 2006. — C. 45—45. — DOI: 10.1109/ICVS.2006.3.

ПРИЛОЖЕНИЕ А

Программный код файла FisheyeDewarper.hpp

```
1 #define _USE_MATH_DEFINES
2 #include "opencv2/core.hpp"
3 #include "opencv2/imgproc.hpp"
4 #include <iostream>
5 #include <vector>
6 #include <numbers>
7 #include <math.h>
8
9 #define SCARAMUZZA      10
10 #define ATAN          20
11 #define REV_SCARAMUZZA  30
12
13 /* TODO: -Constructor,
14      -
15 */
16
17 class FisheyeDewarper
18 {
19 private: /* Parameters */
20     const double PI = M_PI;
21     float xFov;           // output image fov
22     float yFov;           // or 16:9 equivalent
23     cv::Size oldSize;      // input image size
24     cv::Size newSize;      // output image size
25     /* RPY angles for the world point rotator*/
26     float yaw;
27     float pitch;
28     float roll;
29
30 private: /* Data */
31     double errorsum;
32     /* Intrinsics */
33     std::vector<double> polynom;    // Scaramuzza model
                                         coefficients
```

```

34     cv::Vec2d centerOffset;           // Distortion center
35     cv::Matx22d stretchMatrix;       // Stretch matrix
36     double lambda;                 // Scale factor
37     /* Structures */
38     cv::Mat map1;                  // x map
39     cv::Mat map2;                  // y map
40     std::vector<cv::Point> frameBorder; //border to draw on the
41                               original image
42
42 private: /* Internal functions */
43     void createMaps();
44     /* Transformations */
45     void toCenter(cv::Point& cornerPixel, cv::Size imageSize);
46     void toCorner(cv::Point& centerPixel, cv::Size imageSize);
47     cv::Mat rotatePoint(cv::Mat worldPoint);
48     /* Projection functions */
49     cv::Point2f projectWorldToFisheyeAtan(cv::Mat worldPoint);
50     cv::Point2d projectWorldToFisheye(cv::Mat worldPoint);
51     cv::Point2f projectWorldToPinhole(cv::Mat cameraCoords);
52     cv::Mat projectFisheyeToWorld(cv::Point pixel);
53     cv::Mat projectPinholeToWorld(cv::Point pixel);
54     cv::Point reverseScaramuzza(cv::Point pixel); // Fisheye to
55                               Pinhole
55     /* Tools */
56     void fillMapsScaramuzza();
57     void fillMapsRevScaramuzza();
58     void fillMapsAtan();
59     void setFovWide(float wFov);
60
61 public: /* Settings */
62     FisheyeDewarper();
63     void setSize(int oldWidth, int oldHeight, int newWidth, int
64                               newHeight, float wideFov);
64     void setSize(cv::Size oldsize, cv::Size newsize, float wideFov)
65                               ;
65     void setIntrinsics(double coeffs[4], cv::Vec2d centerOffset, cv
66                               ::Matx22d stretchMatrix, double scaleFactor);

```

```
66 void setIntrinsics(double a1, double a2, double a3, double a4,
67                     cv::Vec2d centerOffset, cv::Matx22d stretchMatrix, double
68                     scaleFactor);
69 void setRpy(float yaw, float pitch, float roll);
70
71 public: /* */
72 void fillMaps(int mode);
73 cv::Mat dewrapImage(cv::Mat inputImage);
74
75 };
```

ПРИЛОЖЕНИЕ Б

Программный код файла FisheyeDewarper.cpp

```
1 #include "FisheyeDewarper.hpp"
2
3 FisheyeDewarper::FisheyeDewarper()
4 {
5
6 }
7
8 void FisheyeDewarper::createMaps()
9 {
10    map1 = cv::Mat(oldSize, CV_32FC1, float(0));
11    map2 = cv::Mat(oldSize, CV_32FC1, float(0));
12 }
13
14 void FisheyeDewarper::setSize(int oldWidth, int oldHeight, int
15                               newWidth, int newHeight, float wideFov)
16 {
17     oldSize.width = oldWidth;
18     oldSize.height = oldHeight;
19     newSize.width = newWidth;
20     newSize.height = newHeight;
21     createMaps();
22     setFovWide(wideFov);
23 }
24 void FisheyeDewarper::setSize(cv::Size oldsize, cv::Size newsize,
25                               float wideFov)
26 {
27     oldSize = oldsize;
28     newSize = newsize;
29     createMaps();
30     setFovWide(wideFov);
31 }
32 void FisheyeDewarper::setFovWide(float wFov)
```

```

33  {
34      xFov = wFov;
35      yFov = wFov * newSize.height / newSize.width;           // *
36                  9/16
36
37      // std::cout << "FOV, x: " << xFov << " FOV, y: " << yFov <<
38      std::endl;
38 }
39
40 void FisheyeDewarper::setIntrinsics(double coeffs[4], cv::Vec2d
41     centerOffset, cv::Matx22d stretchMatrix, double scaleFactor)
41 {
42     polynom.assign(coeffs, coeffs+4);           // treats both values
43                 as pointers
43
44     this->centerOffset = centerOffset;
45     this->stretchMatrix = stretchMatrix;
46     this->lambda = scaleFactor;
47 }
48
49 void FisheyeDewarper::setIntrinsics(double a1, double a2, double
50     a3, double a4, cv::Vec2d centerOffset, cv::Matx22d
51     stretchMatrix, double scaleFactor)
51 {
52     polynom.clear();
53     polynom.push_back(a1);
54     polynom.push_back(a2);
55     polynom.push_back(a3);
56     polynom.push_back(a4);
56
57     this->centerOffset = centerOffset;
58     this->stretchMatrix = stretchMatrix;
59     this->lambda = scaleFactor;
60 }
61
62 void FisheyeDewarper::setRpy(float yaw, float pitch, float roll)
63 {

```

```

64     this->yaw    =      yaw * PI / 180;
65     this->pitch = -pitch * PI / 180;
66     this->roll   =      roll * PI / 180;
67
68     // TODO: fillMaps() after angle update. Or should it be
69     // handled manually?
70
71 std::vector<cv::Point> FisheyeDewarper::getBorder() {
72     return this->frameBorder;
73 }
74
75 cv::Point2f FisheyeDewarper::projectWorldToFisheyeAtan(cv::Mat
76     worldPoint)
77 {
78     float wx = worldPoint.at<float>(0);
79     float wy = worldPoint.at<float>(1);
80     float wz = worldPoint.at<float>(2);
81     // sqrt destroys signs so I remember them here
82     int8_t xSign = 1, ySign = 1;
83     if (wx == 0) wx += 0.0001;
84     else if (wx < 0) xSign = -1;
85     if (wy == 0) wy += 0.0001;
86     else if (wy < 0) ySign = -1;
87     if (wz == 0) wz += 0.0001;
88     // fisheye focus
89     double xFocus = newSize.width / PI;
90     double yFocus = newSize.height / PI;
91     cv::Point projectionPoint(newSize.width / 2, newSize.height /
92         2);           // initial value set to the image corner
93
94     // calculate the point location on fisheye image in central
95     // coordinates
96     projectionPoint.x = xSign * xFocus * atan(sqrt(wx * wx + wy *
97         wy) / wz)
98         / sqrt((wy * wy) / (wx * wx) + 1);

```

```

95     projectionPoint.y = ySign * yFocus * atan(sqrt(wx * wx + wy *
96                                         wy) / wz)
97                                         / sqrt((wx * wx) / (wy * wy) + 1);
98
99     // convert to corner coordinates
100    projectionPoint.x = projectionPoint.x + newSize.width / 2;
101    projectionPoint.y = -projectionPoint.y + newSize.height / 2;
102
103    return projectionPoint;
104
105 cv::Point2f FisheyeDewarper::projectWorldToPinhole(cv::Mat
106   cameraCoords)
107 {
108     double xFovRad = xFov * PI / 180;
109     double yFovRad = yFov * PI / 180;
110     double xPinholeFocus = newSize.width / (2 * tan(xFovRad / 2)
111           );
112     double yPinholeFocus = newSize.height / (2 * tan(yFovRad / 2)
113           );
114
115     float cx = cameraCoords.at<float>(0);
116     float cy = cameraCoords.at<float>(1);
117     float cz = cameraCoords.at<float>(2);
118
119     cv::Point2f pinholePoint;
120     pinholePoint.x = xPinholeFocus * cx / cz;
121     pinholePoint.y = yPinholeFocus * cy / cz;
122
123     return pinholePoint;
124 }
125
126 cv::Mat FisheyeDewarper::projectPinholeToWorld(cv::Point pixel)
127 {
128     toCenter(pixel, newSize);
129
130     return pixel;
131 }
```

```

127     float cz = 200.0; //  

128         doesn't really affect much  

129     double xFovRad = xFov * PI / 180;  

130     double yFovRad = yFov * PI / 180;  

131     double xPinholeFocus = newSize.width / (2 * tan(xFovRad / 2))  

132         ;  

133     double yPinholeFocus = newSize.height / (2 * tan(yFovRad / 2))  

134         );  

135  

136     cv::Mat cameraCoords(1, 3, CV_32F, float(0));  

137     cameraCoords.at<float>(0) = pixel.x * cz / xPinholeFocus;  

138     cameraCoords.at<float>(1) = pixel.y * cz / yPinholeFocus;  

139     cameraCoords.at<float>(2) = cz;  

140     //std::cout << xPinholeFocus << " / " << yPinholeFocus << std  

141         ::endl;  

142     return cameraCoords;  

143 }
144  

145 cv::Mat FisheyeDewarper::projectFisheyeToWorld(cv::Point pixel)  

146 {  

147     cv::Vec2d undistPixel = stretchMatrix * (cv::Vec2d(pixel.x,  

148         pixel.y) - centerOffset); // TODO: delete  

149     //std::cout << stretchMatrix << " / " << cv::Vec2d(pixel.x,  

150         pixel.y) - centerOffset << " / " << lambda * undistPixel  

151         [0] << std::endl;  

152     cv::Mat cameraCoords(1, 3, CV_32F, float(0));  

153     double rho = norm(pixel); // sqrt ( x^2 + y^2 )  

154     cameraCoords.at<float>(0) = lambda * undistPixel[0];  

155     cameraCoords.at<float>(1) = lambda * undistPixel[1];  

156     cameraCoords.at<float>(2) = lambda * (polynom[0] + polynom[1]  

157         * pow(rho, 2) + polynom[2] * pow(rho, 3) + polynom[3] *  

158         pow(rho, 4));  

159  

160     return rotatePoint(cameraCoords);  

161 }
162  

163

```

```

154 cv::Point2d FisheyeDewarper::projectWorldToFisheye(cv::Mat
    worldPoint)
155 {
156     double X = worldPoint.at<float>(0);
157     double Y = worldPoint.at<float>(1);
158     double Z = worldPoint.at<float>(2);
159     double phi = atan2(Z, sqrt(X * X + Y * Y));
160     double rho = 0;
161     double error = 1;
162
163     int iter = 0;
164     do
165     {
166         double R = polynom[0];           // R = f(rho)
167         for (int i = 1; i < polynom.size(); i++)
168         {
169             R += polynom[i] * pow(rho, i+1);
170             //std::cout << "R: " << R << std::endl;
171         }
172
173         error = atan2(R, rho) - phi;
174         iter++;
175         rho = rho + 200*error;
176         //std::cout << "It " << iter << " Point: " << worldPoint
177         //<< " / Rho: " << rho << " f(Rho): " << 424242 <<
178         //Error: " << error << std::endl;
179     } while (std::abs(error) > 0.005 && iter < 100);
180     errorsum += error;
181
182     lambda = sqrt(X * X + Y * Y) / rho;
183     double u = X / lambda;
184     double v = Y / lambda;
185
186     cv::Point fypixel( stretchMatrix * cv::Vec2d(u, v) +
187                         centerOffset );           // technically could do toCorner's
188                         // job, but I'll keep it simple for now
189     toCorner(fypixel, oldSize);

```

```

186     return fypixel;
187 }
188
189 cv::Mat FisheyeDewarper::rotatePoint (cv::Mat worldPoint)
190 {
191     cv::Mat rotZ(cv::Matx33f(1, 0, 0,
192                             0, cos(yaw), sin(yaw),
193                             0, -sin(yaw), cos(yaw)));
194     cv::Mat rotX(cv::Matx33f(cos(pitch), 0, -sin(pitch),
195                             0, 1, 0,
196                             sin(pitch), 0, cos(pitch)));
197     cv::Mat rotY(cv::Matx33f(cos(roll), -sin(roll), 0,
198                             sin(roll), cos(roll), 0,
199                             0, 0, 1));
200     return worldPoint * rotY * rotZ * rotX;           // calib3d/
201                         utils proposes this order
202 }
203
204 // converting corner coordinates to the center ones
205 void FisheyeDewarper::toCenter(cv::Point& cornerPixel, cv::Size
206                                 imagesize)
207 {
208     cornerPixel.x = cornerPixel.x - imagesize.width / 2;
209     cornerPixel.y = -cornerPixel.y + imagesize.height / 2;
210 }
211
212 // converting center coordinates to the corner ones
213 void FisheyeDewarper::toCorner(cv::Point& centerPixel, cv::Size
214                                 imagesize)
215 {
216     centerPixel.x = centerPixel.x + imagesize.width / 2;
217     centerPixel.y = -centerPixel.y + imagesize.height / 2;
218 }
219
220 void FisheyeDewarper::fillMaps(int mode)
221 {
222     createMaps();

```

```

220     frameBorder.clear();
221
222     if (mode == SCARAMUZZA) fillMapsScaramuzza();
223     if (mode == REV_SCARAMUZZA) fillMapsRevScaramuzza();
224     if (mode == ATAN)         fillMapsAtan();
225
226 }
227
228 void FisheyeDewarper::fillMapsScaramuzza()
229 {
230     for (int i = 0; i < newSize.width; i++)
231     {
232         for (int j = 0; j < newSize.height; j++)
233         {
234             cv::Mat worldPoint = projectPinholeToWorld(cv::Point(
235                 i, j));
236             worldPoint = rotatePoint(worldPoint);
237             cv::Point distPoint = projectWorldToFisheye(
238                 worldPoint);
239
240             if (distPoint.x > oldSize.width - 1 || distPoint.x <
241                 0 ||
242                 distPoint.y > oldSize.height - 1 || distPoint.y <
243                     0)
244             {
245                 continue; // skips out of border points
246             }
247
248             // save distorted edge of the frame
249             if (((j == 0 || j == newSize.height - 1) && i % 100
250                  == 0) ||
251                 ((i == 0 || i == newSize.width - 1) && j % 100 ==
252                     0))
253             {
254                 frameBorder.push_back(cv::Point(distPoint.y,
255                     distPoint.x));
256             }

```

```

250
251     //map1.at<float>(distPoint.x, distPoint.y) = j;
252     //map2.at<float>(distPoint.x, distPoint.y) = i;
253     map1.at<float>(i, j) = distPoint.y;
254     map2.at<float>(i, j) = distPoint.x;
255 }
256 if (i % 100 == 0) std::cout << "Column N" << i << std::endl;
257 }
258 std::cout << "Avg. error: " << errorsum / (newSize.area()) << std::endl;
259 }
260
261 void FisheyeDewarper::fillMapsRevScaramuzza()
262 {
263     for (int i = 0; i < oldSize.width; i++)
264     {
265         for (int j = 0; j < oldSize.height; j++)
266         {
267             cv::Point distPoint = reverseScaramuzza(cv::Point(i,
268                                         j));
269
270             if (distPoint.x > newSize.width - 1 || distPoint.x <
271                 0 ||
272                 distPoint.y > newSize.height - 1 || distPoint.y <
273                 0)
274             {
275                 continue; // skips out of border points
276             }
277
278             // save distorted edge of the frame
279             if (((j == 0 || j == oldSize.height - 1) && i % 100
280                  == 0) ||
281                 ((i == 0 || i == oldSize.width - 1) && j % 100 ==
282                  0))
283             {

```

```

279         frameBorder.push_back(cv::Point(distPoint.y,
280                               distPoint.x));
281
282         map1.at<float>(i, j) = distPoint.y;
283         map2.at<float>(i, j) = distPoint.x;
284     }
285     if (i%10==0) std::cout << "Column N" << i << std::endl;
286 }
287 std::cout << "Avg. error: " << errorsum / (1080 * 1080) <<
288             std::endl;           // HACK
289
290 void FisheyeDewarper::fillMapsAtan()
291 {
292     for (int i = 0; i < newSize.width; i++)
293     {
294         for (int j = 0; j < newSize.height; j++)
295         {
296             cv::Point distPoint = projectWorldToFisheye(
297                         projectPinholeToWorld(cv::Point(i, j)));
298
299             if (distPoint.x > oldSize.width - 1 || distPoint.x <
300                 0 ||
301                 distPoint.y > oldSize.height - 1 || distPoint.y <
302                 0)
303             {
304                 continue; // skips out of border points
305             }
306
307             // save distorted edge of the frame
308             if (((j == 0 || j == newSize.height - 1) && i % 100
309                  == 0) ||
310                 ((i == 0 || i == newSize.width - 1) && j % 100 ==
311                  0))
312             {

```

```
308         frameBorder.push_back(cv::Point(distPoint.y,
309                               distPoint.x));
310
311         map1.at<float>(i, j) = distPoint.y;
312         map2.at<float>(i, j) = distPoint.x;
313     }
314 }
315 }
316
317 cv::Mat FisheyeDewarper::dewrapImage(cv::Mat inputImage)
318 {
319     cv::Mat remapped(newSize, CV_8UC3, cv::Scalar(0, 0, 0));
320     cv::remap(inputImage, remapped, map1, map2, cv::INTER_CUBIC,
321               cv::BORDER_CONSTANT);
322     return remapped;
323 }
```

ПРИЛОЖЕНИЕ В

Программный код файла unity_plugin.cpp

```
1 //#include "stdafx.h"
2 #include <opencv2/opencv.hpp>
3 #include <opencv2/core/ocl.hpp>
4 #include "surroundView.h"
5 #include <string>
6 #include "../cpp_project/FisheyeDewarper.cpp"
7
8 #define CAMERA_REGULAR 0
9 #define CAMERA_FISHEYE 1
10
11 using namespace std;
12
13 extern "C"
14 {
15     //time's stuff
16     int startTime = 0;
17     int stopTime = 0;
18     int runTime = 0;
19
20     //variables and objects for image transmition
21     vector<Color32*> rawData; //raw data from virtual camera
22     vector<cv::Mat> frames; // all frames
23     vector<cv::cuda::GpuMat> framesGpu;
24     vector<string> framesNames; //and their names
25     cv::ocl::Context context;
26     cv::ocl::Device device;
27
28     bool gotFrames = false;
29     bool isShowed = false;
30
31     cv::Mat output;
32     cv::Mat fisheyeDisparity;
33     cv::Mat regularDisparity;
34
```

```

35 FisheyeDewarper left_dewarper;           // dewarper library
     object
36 FisheyeDewarper right_dewarper;
     // Creating an object of StereoSGBM algorithm
37 //cv::Ptr<cv::StereoBM> stereo;
38 cv::Ptr<cv::StereoSGBM> stereo;
39 // initialize values for StereoSGBM parameters
40
41 int numDisparities = 8;
42 int blockSize = 5;
43 int preFilterType = 1;
44 int preFilterSize = 1;
45 int preFilterCap = 31;
46 int minDisparity = 0;
47 int textureThreshold = 10;
48 int uniquenessRatio = 15;
49 int speckleRange = 0;
50 int speckleWindowSize = 0;
51 int disp12MaxDiff = -1;
52 int dispType = CV_16S;
53
54
55 int camType;
56
57
58 int initialize(int width, int height, int numOfImg, int
     cameraType, int leftRot, int rightRot)
59 {
60     camType = cameraType;
61
62     if (!cv::ocl::haveOpenCL())
63     {
64         return 2;
65     }
66
67     if (!context.create(cv::ocl::Device::TYPE_GPU))
68     {
69         return 3;

```

```

70     }
71
72     for (int i = 0; i < context.ndevices(); i++)
73     {
74         device = context.device(i);
75     }
76
77     // Select the first device
78     cv::ocl::Device(context.device(0));
79
80     if (numOfImg < 1)
81     {
82         return 1;
83     }
84
85     frames.clear();
86     framesNames.clear();
87
88     for (int i = 0; i < numOfImg; i++)
89     {
90         frames.push_back(cv::Mat(height, width, CV_8UC4));
91         framesNames.push_back(to_string(i + 1) + "of" +
92             to_string(numOfImg));
93     }
94
95     rawData.clear();
96
97     for (int i = 0; i < numOfImg; i++)
98     {
99         rawData.push_back(0);
100    }
101
102    if (cameraType == CAMERA_FISHEYE)
103    {
104        // 180 deg: 350.8434, -0.0015, 2.1981 * pow(10, -6),
105        // -3.154 * pow(10, -9)

```

```

104         // 270 deg: 229.3778, -0.0016, 9.737 * pow(10, -7),
105         // -4.2154 * pow(10, -9)
106         left_dewarper.setIntrinsics(350.8434, -0.0015, 2.1981
107             * pow(10, -6), -3.154 * pow(10, -9), cv::Vec2d(0,
108                 0), cv::Matx22d(1, 0, 0, 1), 0.022);           //
109         // 270 deg coeffs
110         left_dewarper.setSize(cv::Size(1080, 1080), cv::Size
111             (540, 540), 90);
112         left_dewarper.setRpy(leftRot, 0, 0);
113         left_dewarper.fillMaps(SCARAMUZZA);
114
115         right_dewarper.setIntrinsics(350.8434, -0.0015,
116             2.1981 * pow(10, -6), -3.154 * pow(10, -9), cv::
117                 Vec2d(0, 0), cv::Matx22d(1, 0, 0, 1), 0.022);
118         right_dewarper.setSize(cv::Size(1080, 1080), cv::Size
119             (540, 540), 90);
120         right_dewarper.setRpy(rightRot, 0, 0);
121         right_dewarper.fillMaps(SCARAMUZZA);
122
123
124         int screenIndex = 0;
125
126         int takeScreenshot(Color32** raw, int width, int height, int
127             numOfCam, bool isShow)
128         {
129             cv::Mat sceen = cv::Mat(height, width, CV_8UC4, raw[
130                 numOfCam]);

```

```

129         cvtColor(sceen, sceen, cv::COLOR_BGRA2RGB);
130         flip(sceen, sceen, 0);
131         screenIndex++;
132         string path = "D:/Work/Coding/Repos/RTC_Practice/
133             fisheye_stereo/data/stereo_img/" + to_string(
134                 screenIndex) + "_shot.jpg";
135         cv::imwrite(path, sceen);
136
137         if (isShow)
138         {
139             imshow("OpenCV Screenshot", sceen);
140         }
141     }
142
143     int takeStereoScreenshot(Color32** raw, int width, int height
144         , int cameraType, int numOfCam1, int numOfCam2, bool
145         isShow)
146     {
147         cv::Mat stereo = cv::Mat(height, 2 * width, CV_8UC4, cv::
148             Scalar(0, 0, 0));
149         cv::Mat cam1 = cv::Mat(height, width, CV_8UC4, raw[
150             numOfCam1]);
151         cv::Mat cam2 = cv::Mat(height, width, CV_8UC4, raw[
152             numOfCam2]);
153
154         cv::Mat de_cam1 = cv::Mat(540, 540, CV_8UC4);
155         cv::Mat de_cam2 = cv::Mat(540, 540, CV_8UC4);
156
157         cvtColor(cam1, cam1, cv::COLOR_BGRA2RGB);
158         flip(cam1, cam1, 0);
159         cvtColor(cam2, cam2, cv::COLOR_BGRA2RGB);
160
161         screenIndex++;
162         string left_path;
163         string right_path;

```

```

159     if (cameraType == CAMERA_REGULAR) {
160         left_path = "D:/Work/Coding/Repos/RTC_Practice/
161             fisheye_stereo/data/stereo_img/" + to_string(
162                 screenIndex) + "_reg_l_shot.jpg";
163         right_path = "D:/Work/Coding/Repos/RTC_Practice/
164             fisheye_stereo/data/stereo_img/" + to_string(
165                 screenIndex) + "_reg_r_shot.jpg";
166         cv::imwrite(left_path, cam1);
167         cv::imwrite(right_path, cam2);
168     }
169     else {
170         //de_cam1 = ;
171         de_cam2 = right_dewarper.dewrapImage(cam2);
172         de_cam1 = left_dewarper.dewrapImage(cam1);
173         left_path = "D:/Work/Coding/Repos/RTC_Practice/
174             fisheye_stereo/data/stereo_img/" + to_string(
175                 screenIndex) + "_fy_l_shot.jpg";
176         right_path = "D:/Work/Coding/Repos/RTC_Practice/
177             fisheye_stereo/data/stereo_img/" + to_string(
178                 screenIndex) + "_fy_r_shot.jpg";
179
180         if (isShow)
181         {
182             imshow("OpenCV Screenshot", cam1);
183         }
184
185         return 0;
186     }
187
188     int getImages(Color32** raw, int width, int height, int
189                 cameraType, int numOfImg, bool isShow, SGBMparams sgbm)
190     {

```

```

185     if (numOfImg < 1)
186     {
187         gotFrames = false;
188         return 1;
189     }
190     else
191     {
192 #pragma omp parallel for
193         for (int i = 0; i < numOfImg; i++)
194         {
195             frames[i] = cv::Mat(height, width, CV_8UC4, raw[i]
196             ]);
197             cvtColor(frames[i], frames[i], cv::COLOR_BGRA2RGB
198             );
199             flip(frames[i], frames[i], 0);
200         }
201         if (cameraType == CAMERA_FISHEYE) {
202             frames[0] = left_dewarper.dewrapImage(frames[0]);
203             // undistort
204             frames[1] = right_dewarper.dewrapImage(frames[1])
205             ;
206         }
207         fillStereoParams(sgbm);
208         cv::Mat disparity = calculateDisparities(frames[0],
209             frames[1], cameraType);
210         if (isShow)
211         {
212             imshow(framesNames[i], frames[i]);
213             isShowed = true;
214         }
215     }
216     else

```

```

217         {
218             if (isShowed)
219             {
220                 for (int i = 0; i < numOfImg; i++)
221                 {
222                     cv::destroyWindow(framesNames[i]);
223                     isShowed = false;
224                 }
225             }
226         }
227
228         gotFrames = true;
229     }
230
231     return 0;
232 }
233
234
235     void processImage(unsigned char* data, int width, int height)
236     {
237         //Convert from RGB to ARGB
238         cv::Mat argb_img;
239         cvtColor(output, argb_img, cv::COLOR_RGB2BGRA);
240         cv::flip(argb_img, argb_img, 0);
241         vector<cv::Mat> bgra;
242         split(argb_img, bgra);
243         swap(bgra[0], bgra[3]);
244         swap(bgra[1], bgra[2]);
245         merge(bgra, argb_img);
246         memcpy(data, argb_img.data, argb_img.total() * argb_img.
247             elemSize());
248     }
249
250     void terminate()
251     {
252         rawData.clear();
253         frames.clear();

```

```

253         cv::destroyAllWindows();
254     }
255 }
256
257 void fillStereoParams(SGBMparams& sgbm)
258 {
259     stereo->setBlockSize(sgbm.blockSize * 2 + 5);
260     stereo->setPreFilterCap(sgbm.preFilterCap);
261     //stereo->setPreFilterSize(sgbm.preFilterSize*2+5);
262     stereo->setP1(sgbm.preFilterSize);
263     stereo->setMinDisparity(sgbm.minDisparity);
264     stereo->setNumDisparities(sgbm.numDisparities * 16);
265     //stereo->setTextureThreshold(sgbm.textureThreshold);
266     stereo->setP2(sgbm.textureThreshold);
267     stereo->setUniquenessRatio(sgbm.uniquenessRatio);
268     stereo->setSpeckleWindowSize(sgbm.speckleWindowSize * 2);
269     stereo->setSpeckleRange(sgbm.speckleRange);
270     stereo->setDisp12MaxDiff(sgbm.disp12MaxDiff);
271     // 2 pass expensive method
272     // stereo->setMode(cv::StereoSGBM::MODE_HH);
273
274 }
275
276 cv::Mat calculateDisparities(cv::Mat leftImage, cv::Mat
277                             rightImage, int cameraType) {
278     cv::Mat disp;
279     // Converting images to grayscale
280     cv::cvtColor(leftImage, leftImage, cv::COLOR_BGR2GRAY);
281     cv::cvtColor(rightImage, rightImage, cv::COLOR_BGR2GRAY);
282
283     // Calculating disparity using the StereoBM algorithm
284     stereo->compute(leftImage, rightImage, disp);
285
286     // Converting disparity values to CV_32F from CV_16S
287     disp.convertTo(disp, CV_32F, 1.0);
288
289     // Scaling down the disparity values and normalizing them

```

```
289     disp = (disp / 16.0f - (float)stereo->getMinDisparity())  
290     / ((float)stereo->getNumDisparities());  
291  
292     if (cameraType == CAMERA_FISHEYE) {  
293         fisheyeDisparity = disp;  
294         imshow("Fisheye disparity", fisheyeDisparity);  
295     }  
296     if (cameraType == CAMERA_REGULAR) {  
297         regularDisparity = disp;  
298         imshow("Regular disparity", regularDisparity);  
299     }  
300     cv::waitKey(1);  
301  
302     return disp;  
303 }
```

ПРИЛОЖЕНИЕ Г

Программный код файла Connector.cs

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using System;
4 using System.Globalization;
5 using System.Linq;
6 using System.Runtime.ExceptionServices;
7 using System.Runtime.InteropServices;
8 using System.Threading;
9 using System.Threading.Tasks;
10 using UnityEngine;
11 using UnityEngine.UI;
12 using MultiThreading;
13
14
15
16 public struct SGBMparams
17 {
18     public byte preFilterSize;
19     public byte preFilterCap;
20     public byte blockSize;
21     public byte minDisparity;
22     public byte numDisparities;
23     public byte textureThreshold;
24     public byte uniquenessRatio;
25     public byte speckleRange;
26     public byte disp12MaxDiff;
27     public byte speckleWindowSize;
28 };
29
30 public class Connector : MonoBehaviour
31 {
32     [Header("Cameras")]
33     public Camera camera1;
34     public int cam1XRot;
```

```

35     public Camera camera2;
36     public int cam2XRot;
37     public Camera camera3;
38     public Camera camera4;
39
40     [Header("Screens")]
41     public GameObject disparityScreen;
42     // for bowl
43     private int dispScreenWidth = 1080;
44     private int dispScreenHeight = 1080;
45     private Texture2D dispScreenTexture;
46     private Color32[] dispScreenPixels;
47     private GCHandle pixelHandle;
48     private IntPtr pixelPtr;
49
50
51     [Header("Sliders")]
52     public Slider preFilterSize;
53     public Slider preFilterCap;
54     public Slider blockSize;
55     public Slider minDisparity;
56     public Slider numDisparities;
57     public Slider textureThreshold;
58     public Slider uniquenessRatio;
59     public Slider speckleRange;
60     public Slider speckleWindowSize;
61     public Slider disp12MaxDiff;
62
63     [Header("Img parameters")]
64     public static int width = 1080;
65     public static int height = 1080;
66     public static bool showImages = false;
67     public bool showPano = false;
68     // Fisheye
69     private Texture2D camTex1;
70     private Texture2D camTex2;
71     // Regular

```

```

72     private Texture2D camTex3;
73     private Texture2D camTex4;
74
75     private SGBMparams sgbm;           // structure to store SGBM
76                           parameter values
76
77     private Rect readingRect;
78     int isOk = 0;
79     public int gap = 40;
80     int actionId = 0;
81
82     bool pano = false;
83     // Threading stuff
84     DisparityCalculator imageProcessingThread;
85     DisparityCalculator regularCameraThread;
86     bool threadStarted = false;
87     bool initFlag = false;
88     bool turn = true;
89
90     // Start is called before the first frame update
91     void Start()
92     {
93         Debug.Log("Start Function");
94         InitTexture();
95         disparityScreen.GetComponent<Renderer>().material.
96             mainTexture = dispScreenTexture;
96         unsafe {
97             //AllocConsole();
98             sgbm = new SGBMparams();
99             imageProcessingThread = new DisparityCalculator(1,
100                 1080, 1080, showImages, cam1XRot, cam2XRot,
101                 pixelPtr);
100             regularCameraThread = new DisparityCalculator(0, 540,
102                 540, showImages, 0, 0, pixelPtr);
101         }
102
103     }

```

```

104     // Update is called once per frame
105     void Update()
106     {
107         // Fisheye cameras
108         camTex1 = CameraToTexture2D(camera1);
109         camTex2 = CameraToTexture2D(camera2);
110         // Regular cameras
111         camTex3 = CameraToTexture2D(camera3);
112         camTex4 = CameraToTexture2D(camera4);
113
114         fillStereoParams();           // fills 'sgbm' structure with
115                                     // slider values
116         actionId = 0;                // Default to update action
117         // TODO: id's are messed up
118         /* Update texture in the world */
119         if (Input.GetKeyUp("[1]")) {
120             actionId = 3;
121             // TODO: World texture renderer
122         }
123         /* Take screenshot */
124         if (Input.GetKeyUp("[2]")) {
125             actionId = 1;
126         }
127         /* Reinitialize */
128         if (Input.GetKeyUp("[3]")) {
129             actionId = 2;
130         }
131         /* Start the second thread */
132         if (!threadStarted)          // image processing thread
133             hasn't been started before
134         {
135             Debug.Log("Starting thread");
136             regularCameraThread.Start();
137             imageProcessingThread.Start();    // start the
138                                         // threads
139             threadStarted = true;

```

```

138         }
139         /* Pass the data to the thread */
140         // idk seems like both threads use the same library
141         // memory
142         if (imageProcessingThread.code == 0 &&
143             regularCameraThread.code == 0 || true)
144         {
145             imageProcessingThread.Update(camTex1.GetPixels32(),
146                                         camTex2.GetPixels32(), sgbm, actionId);
147             regularCameraThread.Update(camTex3.GetPixels32(),
148                                         camTex4.GetPixels32(), sgbm, actionId);
149         }
150
151         /* textures are not erased by the GC automatically */
152         Destroy(camTex1);
153         Destroy(camTex2);
154         Destroy(camTex3);
155         Destroy(camTex4);
156     }
157
158     void OnApplicationQuit()
159     {
160         pixelHandle.Free();
161         imageProcessingThread.Abort();
162         regularCameraThread.Abort();
163     }
164
165     void OnGUI()
166     {
167         GUI.Label(new Rect(0, 0, 100, 100), (1.0f / (Time.
168                                         smoothDeltaTime)).ToString());
169     }
170
171     private Texture2D CameraToTexture2D(Camera camera)
172     {
173         Rect rect;

```

```

170     RenderTexture renderTexture = new RenderTexture(1080,
171         1080, 24);           // TODO: adapt texture for regular and
172         fisheye
173
174     Texture2D screenShot = new Texture2D(540, 540,
175         TextureFormat.ARGB32, false);
176
177     if (camera == camera3)
178     { rect = new Rect(0, height / 2, width / 2, height / 2);
179
180     else if (camera == camera4)
181     { rect = new Rect(width / 2, height / 2, width / 2,
182         height / 2); }
183
184     else           // full frame fusheye
185     {
186
187         rect = new Rect(0, 0, width, height);
188
189         renderTexture = new RenderTexture(width, height, 24);
190
191         screenShot = new Texture2D(width, height,
192             TextureFormat.ARGB32, false);
193
194     }
195
196
197
198     void InitTexture()
199     {

```

```

200     dispScreenTexture = new Texture2D(dispScreenWidth,
201         dispScreenHeight, TextureFormat.ARGB32, false);
202     dispScreenPixels = dispScreenTexture.GetPixels32();
203     //Pin pixel32 array
204     pixelHandle = GCHandle.Alloc(dispScreenPixels,
205         GCHandleType.Pinned);
206     //Get the pinned address
207     pixelPtr = pixelHandle.AddrOfPinnedObject();
208 }
209
210 unsafe void MatToTexture2D()
211 {
212     //Convert Mat to Texture2D
213     processImage(pixelPtr, dispScreenWidth, dispScreenHeight)
214     ;
215     //Update the Texture2D with array updated in C++
216     dispScreenTexture.SetPixels32(dispScreenPixels);
217     dispScreenTexture.Apply();
218 }
219
220 void fillStereoParams()
221 {
222     sgbm.preFilterCap      = (byte)preFilterCap.value;
223     sgbm.preFilterSize     = (byte)preFilterSize.value;
224     sgbm.blockSize         = (byte)blockSize.value;
225     sgbm.minDisparity      = (byte)minDisparity.value;
226     sgbm.numDisparities    = (byte)numDisparities.value;
227     sgbm.textureThreshold = (byte)textureThreshold.value;
228     sgbm.uniquenessRatio   = (byte)uniquenessRatio.value;
229     sgbm.speckleRange       = (byte)speckleRange.value;
230     sgbm.disp12MaxDiff     = (byte)disp12MaxDiff.value;
231 }
232
233 #region dllimport
234
235 [DllImport("unity_plugin", EntryPoint = "initialize")]

```

```

233     unsafe private static extern int initialize(int width, int
234         height, int num, int imageType, int leftRot, int rightRot)
235         ;
236
237
238     [DllImport("unity_plugin", EntryPoint = "terminate")]
239     unsafe private static extern void terminate();
240
241
242     [DllImport("unity_plugin", EntryPoint = "processImage")]
243     unsafe private static extern void processImage(IntPtr data,
244         int width, int height);
245
246     [DllImport("unity_plugin", EntryPoint = "getImages")]
247     unsafe private static extern int getImages(IntPtr raw, int
248         width, int height, int numOfImg, int cameraType, bool
249         isShow, SGBMparams sgbm);
250
251
252     [DllImport("unity_plugin", EntryPoint = "takeScreenshot")]
253     unsafe private static extern int takeScreenshot(IntPtr raw,
254         int width, int height, int numOfCam, bool isShow);
255
256
257     [DllImport("unity_plugin", EntryPoint = "takeStereoScreenshot
258        ")]
259     unsafe private static extern int takeStereoScreenshot(IntPtr
260         raw, int width, int height, int numOfCam1, int numOfCam2,
261         bool isShow);
262
263
264     [DllImport("unity_plugin", EntryPoint = "AllocConsole")]
265     private static extern bool AllocConsole();
266
267     #endregion
268 }

```

ПРИЛОЖЕНИЕ Д

Программный код файла DisparityPlayground.m

```
1
2 SHOW = false;
3 distances = [4 5 6 7.5 10 12.5 15]; % // 13
4 regErrors = [];
5 fyErrors = [];
6
7 for dst = distances
8
9     %% FISHEYE %%
10    fy_e = computePlaneError(newFisheyeStereoParams, dst, "fy",
11        SHOW);
12    fyErrors = [fyErrors fy_e];
13    %% REGULAR %%
14    reg_e = computePlaneError(newRegularStereoParams, dst, "reg",
15        SHOW);
16    regErrors = [regErrors reg_e];
17    disp("Distance ready")
18
19
20
21
22 function [MSE] = computePlaneError(stereoParams, distance, type,
23     show)
24     base_path = "D:\Work\Coding\Repos\RTC_Practice\fisheye_stereo
25         \data\stereo_img\compar\plane" + string(distance) + "m\";
26
27     targetDistance = distance;
28     target_roi = [-0.4 0.6 -0.6 0.46 targetDistance/10-0.05
29         targetDistance/10+0.05];
30
31     targetParamsVector = [0, 0, 1, -targetDistance/10]; %
32         normal + distance
```

```

29     ref_model = planeModel(targetParamsVector);
30
31     imgRight = base_path + type + "_r_shot.jpg";
32     imgLeft = base_path + type + "_l_shot.jpg";
33
34     lImage = imread(imgLeft);
35     rImage = imread(imgRight);
36
37     [frameLeftRect, frameRightRect] = rectifyStereoImages(lImage,
38               rImage, stereoParams);
39
40     %figure;
41     %imshow(stereoAnaglyph(frameLeftRect, frameRightRect));
42     %title('Rectified Video Frames');
43
44     frameLeftGray = rgb2gray(frameLeftRect);
45     frameRightGray = rgb2gray(frameRightRect);
46
47     disparityMapReg = disparitySGM(frameLeftGray, frameRightGray)
48         ;
49     % figure;
50     % imshow(disparityMapReg, [0, 64]);
51     % title('Disparity Map');
52     % colormap jet
53     % colorbar
54
55     points3Dreg = reconstructScene(disparityMapReg, stereoParams)
56         ;
57     points3Dreg = points3Dreg ./ 1000;
58     ptCloud = pointCloud(points3Dreg, 'Color', frameLeftRect);
59     indices = findPointsInROI(ptCloud, target_roi);
60     ptCloud = select(ptCloud, indices);
61
62     %
63     MSE = findMSE(ptCloud, ref_model);
64
65     if (show)
66         disp(type"ERROR: ")

```

```

63     disp(MSE)
64     % Visualize the point cloud
65     figure('Name',type' depth')
66     pcshow(ptCloud);
67     hold on
68     plot(ref_model, 'color', 'white')
69     hold off
70   end
71 end
72
73 function [MSE] = findMSE(pt_cloud, plane_model)
74     errorSum = 0.0;
75     for elm = 1:pt_cloud.Count
76         pnt = pt_cloud.Location(elm,:);
77         pnt_error = findSquareError(pnt, plane_model);
78         errorSum = errorSum + pnt_error;
79
80
81     end
82
83     MSE = sqrt(errorSum/pt_cloud.Count);
84 end
85
86 function [error] = findSquareError(point, plane)
87     planePoint = [0, 0, -plane.Parameters(4)]; % plane center
88     PQ = point - planePoint;
89     error = (dot(PQ, plane.Normal)*10 )^2;        % squared
90     difference
91 end
92
93 function createfigure(X1, YMatrix1)
94 %CREATEFIGURE(X1, YMatrix1)
95 % X1: vector of x data
96 % YMATRIX1: matrix of y data
97 % Auto-generated by MATLAB on 16-Jan-2022 02:26:31
98

```

```

99 % Create figure
100 figure1 = figure;
101
102 % Create axes
103 axes1 = axes('Parent',figure1);
104 hold(axes1,'on');
105
106 % Create multiple lines using matrix input to plot
107 plot1 = plot(X1,YMatrix1,'Marker','square');
108 set(plot1(1),'DisplayName','oooooooooooooo" oooooooo');
109 set(plot1(2),'DisplayName','oooooooooooooo oooooooo');
110
111 % Create ylabel
112 ylabel('oooooooooooooo oooooooo', 'y');
113
114 % Create xlabel
115 xlabel('oooooooooooooo oo oooooooo', 'x');
116
117 % Uncomment the following line to preserve the X-limits of the
      axes
118 xlim(axes1, [-0.000209902368405811 16.0517554642163]);
119 % Uncomment the following line to preserve the Y-limits of the
      axes
120 ylim(axes1, [-3.8322756199846e-05 0.35351683918712]);
121 box(axes1,'on');
122 % Set the remaining axes properties
123 set(axes1,'XGrid','on','YGrid','on');
124 % Create legend
125 legend1 = legend(axes1,'show');
126 set(legend1, ...
127     'Position',[0.172415836709068 0.839227796053986
                  0.262158950099176 0.0602310215285902]);
128 end

```