Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования

РАБОТА ПРОВЕРЕНА
Рецензент, к.т.н.,
доцент кафедры ИТЭ,
_____/ Е.М. Сартасов
« ____ » _____2018 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ПМиП,
д.ф.-м.н.
_____/А.А.Замышляева
« ____ » _____2018 г.

Оценивание пространственного положения и ориентации жесткого объекта по данным с RGB-D камеры и инерционных датчиков (IMU)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ВЫПУСКНОЙ
КВАЛИФИКАЦИОННОЙ РАБОТЕ МАГИСТРА
ЮУрГУ–01.04.02.2018.187.ПЗ ВКР

Руководитель работы, к.п.н.,
доцент кафедры ПмиП
_____/ А.Ю. Эвнин
«____»_____2018 г.

Автор работы
студент группы ЕТ-222
_____/ Р.Г. Хайруллин
«____»_____2018 г.

Нормоконтролер, к.э.н.,
доцент кафедры ПМиП
_____/ Д.А. Дрозин
«____»_____2018 г.

Челябинск, 2018

АННОТАЦИЯ

Хайруллин Р.Г. Оценивание пространственного положения и ориентации жесткого объекта по данным с RGB-D камеры и инерционных датчиков (IMU). – Челябинск: ЮУрГУ, ЕТ-222, 80 с., 32 ил., 1 табл., библиогр. список – 50 наим., 1 прил.

Оценивание положения и ориентации объекта в реальном времени является сложной и важной задачей компьютерного зрения. Эта задача возникает во многих областях, таких как робототехника, дополненная и виртуальная реальность и др. В данной работе предложен метод нахождения параметров положения объекта в пространстве на основе данных с RGB-D сенсора с использованием инерционных датчиков (IMU). Продемонстрировано применение алгоритма к решению задачи трекинга шлема виртуальной реальности. Для улучшения трекинга в качестве дополнительного фактора были использованы позиции ключевых точек скелета.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 7
1. МЕТОДЫ ТРЕКИНГА И ОЦЕНКИ ПАРАМЕТРОВ ОБЪЕКТОВ 8
1.1. Методы детекции 8
1.2. Методы трекинга 11
1.3. Методы оценивания параметров модели 13
Выводы по разделу 15
2. МАТЕМАТИЧЕСКАЯ МОДЕЛЬ КАМЕРЫ И ОБЪЕКТА 16
2.1. RGB-D сенсор, карта глубины 16
2.2. Модель камеры 17
2.2.1. Дисторсия 18
2.2.2. Калибровка камеры 19
2.3. Алгоритм калибровки 20
2.3.1. Поиск паттерна 20
2.3.2. Детекция размытых изображений 20
2.3.3. Обеспечение покрытия пространства поз 21
2.3.4. Устранение дисторсии 22
2.4. Модель жесткого объекта 22
Выводы по разделу 23
3. ОЦЕНИВАНИЕ ПАРАМЕТРОВ ПО RGB-D ДАННЫМ 25
3.1. Поиск особых точек 25
3.1.1. Поиск шахматных углов. 27
3.2. Выбор особых точек 29
3.3. Поиск подграфа максимального веса 31
3.4. Субпиксельное уточнение позиций 33
3.4.1. Алгоритм Mean-Shift 33
3.5. Дисперсия поиска особых точек 35
3.6. Оптимизация ошибки репроекции 36
3.7. Двухмодальность оценки 37
3.8. Аномалия фронтальной позы 38
3.9. Карта глубины 39
Выводы по разделу 42
4. ИНЕРЦИОННЫЕ ДАТЧИКИ (IMU) 44
4.1. Акселерометр 44
4.2. Интегрирование акселерометра 45
4.3. Гироскоп 47

4.4. Интегрирование гироскопа ... 47
4.5. Магнитометр ... 48
4.6. Применение датчиков ... 48
4.7. Трекинг шлема виртуальной реальности ... 49
4.7.1. Постановка задачи ... 50
4.7.2. Задача автокалибровки ... 52
4.7.3. Частный случай задачи автокалибровки ... 55
Выводы по разделу ... 59
5. АНАЛИЗ РЕЗУЛЬТАТОВ РАБОТЫ ... 60
5.1. Дисперсия оценки вращения ... 60
5.2. Дисперсия оценки смещения ... 61
5.3. Оценка производительности программы ... 64
Выводы по разделу ... 65
ЗАКЛЮЧЕНИЕ ... 66
БИБЛИОГРАФИЧЕСКИЙ СПИСОК ... 69
ПРИЛОЖЕНИЕ 1. КОД ПРОГРАММЫ ... 73

ВВЕДЕНИЕ

Распознавание положения и конфигурации объекта в реальном времени является сложной и важной задачей компьютерного зрения. Эта задача возникает во многих областях, таких как робототехника, дополненная и виртуальная реальность и др.

В работе исследуется задача нахождения положения и ориентации жесткой модели по наблюдаемым данным с сенсора, содержащего RGB-D камеру, а также данным с инерционных датчиков (IMU).

Цель работы – по поступающим с сенсора, в области видимости которого находится объект, последовательности кадров, оценить параметры позы жесткой модели (расположение и ориентацию), соответствующей этому объекту, на каждом кадре. Эту оценку необходимо применить в комбинации с данными инерционных датчиков IMU для улучшения точности и устойчивости.

Для достижения цели необходимо решить следующие задачи.

1. Параметризация модели.
2. Поиск особых точек с субпиксельной точностью.
3. Детекция объекта с помощью поиска подграфа заданной структуры с максимальным весом.
4. Задача минимизации ошибки репроекции.
5. Применение карты глубины для решения проблем фронтального положения и двухмодальности.
6. Применение алгоритма для трекинга шлема виртуальной реальности.
7. Использование скелетной модели (задача автокалибровки).

В работе предложен метод нахождения параметров модели. Продемонстрировано применение к решению задачи трекинга шлема виртуальной реальности.

1. МЕТОДЫ ТРЕКИНГА И ОЦЕНКИ ПАРАМЕТРОВ ОБЪЕКТОВ

Задача оценивания параметров модели по последовательности изображений (видео) как правило делится на два этапа:

1) поиск объекта на изображении с последующим трекингом;

2) непосредственное оценивание параметров модели объекта, используя данные детекции и трекинга.

1.1. Методы детекции

Первым шагом в процессе трекинга является идентификация интересующих объектов на последовательности кадров и кластеризация пикселей этих объектов. Так как движущиеся объекты как правило являются первичным источником информации, большинство методов фокусируются на детекции таких объектов. Далее приведено описание различных методов.

1.1.1. Разность кадров

Наличие движущихся объектов определяется вычислением разности между двумя соседними кадрами (рисунок 1.1).

Рисунок 1.1 – Разность кадров

Эти вычисления простые и их легко реализовать. Имеется высокая возможность адаптации для различных динамических сцен, но, в общем случае сложно получить полный контур движущегося объекта, вследствие чего не удается получить результат детекции с высокой точностью [1].

1.1.2. Оптический поток

Метод оптического потока [2] предполагается вычисление оптического поля изменения изображения с последующей кластеризацией распределения поля на изображении (рисунок 1.2).

Рисунок 1.2 – Оптический поток

Этот метод позволяет получить полную информацию о движении и лучше дифференцировать движущийся объект от фона. Однако, большое количество вычислений и слабая устойчивость к шуму делают этот метод неприменимым для задач трекинга в реальном времени.

1.1.3. Вычитание фона

Первый шаг вычитания фона (рисунок 1.3) – построение его модели. Это главная часть алгоритма вычитания фона.

Рисунок 1.3 – Вычитание фона

Модель фона должна быть достаточно чувствительной, чтобы распознать движущиеся объекты [3]. Моделирование фона должно дать референсную модель. Эта модель используется при вычитании фона, при котором каждый кадр сопоставляется с ней, чтобы получить возможные вариации. Различия между референсной моделью и текущим кадром сигнализирует о наличии движущихся объектов. В настоящее время для реализации модели фона широко используются медианный и усредняющий фильтры. Этот алгоритм позволяет получить наиболее полную информацию об объектах в случае, когда фон известен. Как описано в [4], вычитание фона выполняется с помощью двух подходов.

1. Рекурсивный алгоритм.

Рекурсивные методы [4][5] не хранят буфер для оценки фона. Вместо этого, они рекурсивно обновляют модель фона, основываясь на каждом входном кадре. Как результат, входные кадры, далекие во времени, могут давать эффект при оценке текущей модели. В сравнении с нерекурсивными методами они требуют меньше памяти, но любая ошибка в модели фона может сохраняться на более долгом промежутке времени. Эти методы включают различные техники, такие как приближенная медиана, адаптивный фон, смесь гауссиан.

2. Нерекурсивный алгоритм.

Нерекурсивные методы [4][5] для оценки фона используют подход скользящего окна. Хранится буфер из нескольких предыдущих кадров и оценивается изображение фона, основанное на временном изменении каждого пикселя внутри буфера. Эти методы хорошо адаптируются, т.к. они не зависят от кадров, не попадающих в буфер. В другой стороны, требования по памяти могут быть значительно выше, т.к. чтобы справиться с медленным движением требуется большой буфер.

В таблице 1.1 приведено сравнение методов.

Таблица 1.1.

Сравнение методов детекции и трекинга

Методы		Точность	Время работы	Плюсы и минусы
Вычитание фона	Смесь гауссиан	Средняя	Среднее	+ Низкое требование по памяти
				- Не справляется с мультимодальным фоном
	Приближенная медиана	Средне-низкая	Среднее	+ Не требует сэмплинг кадров
				- Требует хранение буфера с последними значениями пикселей
Оптический поток		Средняя	Высокое	+ Позволяет получить полную информацию о движении объектов
				- Требует большого количества вычислений

Вычитание кадров	Высокая	Средне-низкое	+ Наиболее простой метод. Хорошо работает для статического фона
			- Требует наличие фона без объектов

1.2. Методы трекинга

Трекинг – задача определения траектории движения объекта внутри сцены на изображении (рисунок 1.4).

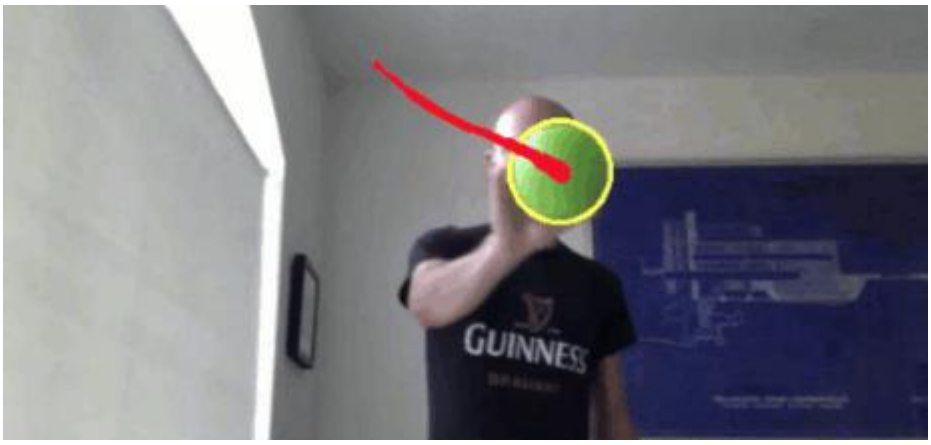


Рисунок 1.4 – Трекинг объекта

Цель трекинга – определить движение объекта во времени и найти его позицию на каждом кадре последовательности [6]. Согласно [3], трекинг объекта может быть классифицирован на точечный трекинг, методы, основанные на оценке функции ядра и трекинг контуров. Для примера, точечные трекеры выполняют детекцию на каждом кадре, тогда как трекеры, основанные на геометрических свойствах, функции ядра или контурах требуют детекции только когда объект впервые появляется в сцене. Как описано в [3], методы трекинга могут быть разделены на следующие категории.

1.2.1. Точечный трекинг

В структуре изображения во время трекинга движущиеся объекты представляются их особыми точками (рисунок 1.5).

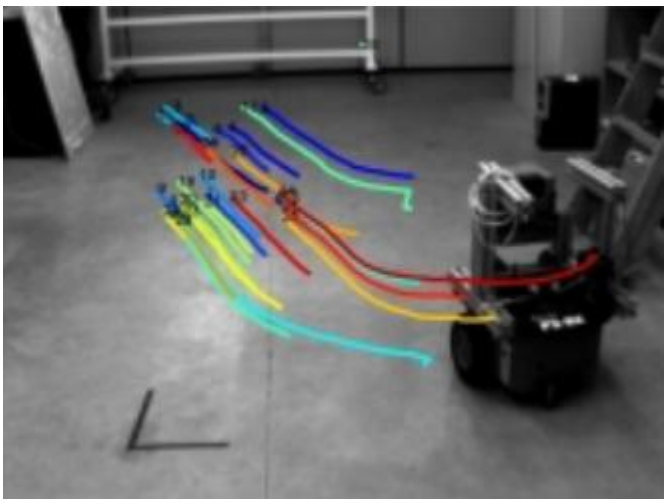


Рисунок 1.5 – Трекинг особых точек

Трекинг точек [3] – сложная задача, особенно в случае перекрытия и при ложных детекциях объекта. Распознавание может быть сделано относительно просто с помощью отсечения по порогу при идентификации этих точек.

1.2.2. Трекинг ядра

Трекинг ядра [7] обычно выполняется с помощью вычисления движущегося объекта, который представляется регионом, от кадра к кадру, например, эллипсом (рисунок 1.6).

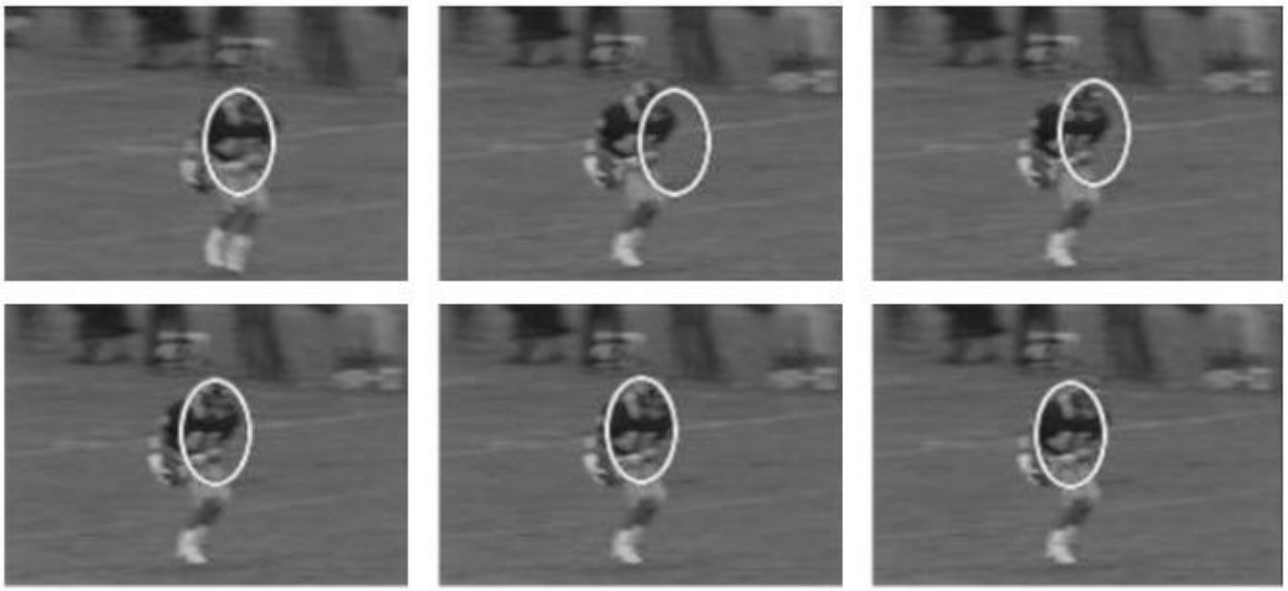


Рисунок 1.6 – Трекинг ядра

Движение объекта представляется в виде параметризованного отображения, такого как смещение, конформное или аффинное преобразование.

Эти алгоритмы отличаются с точки зрения использования представления наличия объекта, количества объектов и метода, который используется для аппроксимации движения объекта. В настоящее время, для представления объекта, наи-

более часто используется геометрическая форма. Одним из ограничений является то, что часть объекта может быть снаружи определенной формы, тогда как куски фона могут быть внутри. Это может быть обнаружено для жестких и деформируемых объектов. Основными методами являются простой поиск шаблона [7], метод скользящего центра, метод опорных векторов, слоевой трекинг [8].

1.2.3. Трекинг силуэта

Некоторые объекты (например, руки, пальцы, плечи) имеют сложную форму, которая не может быть корректно определена с помощью простых геометрических форм. Методы, основанные на силуэтах [7] позволяют получить точное описание формы для таких объектов (рисунок 1.7).

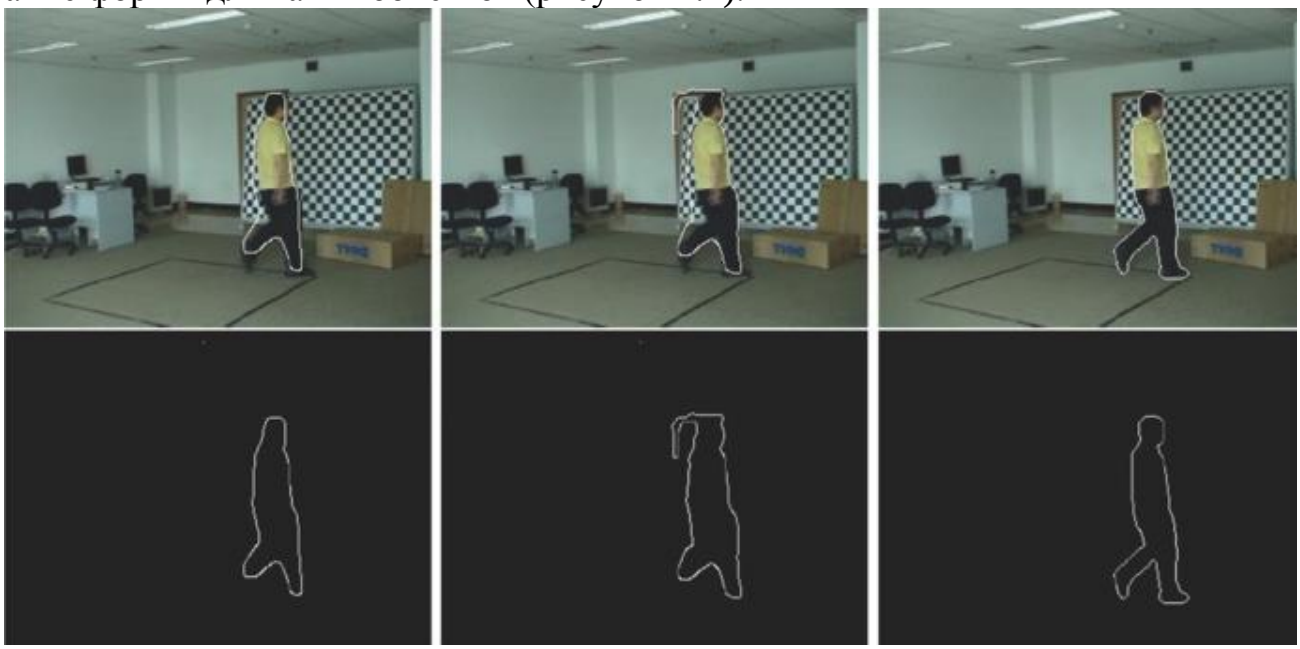


Рисунок 1.7 – Трекинг силуэта

Цель этих методов – найти область объекта на каждом кадре в терминах модели объекта, сгенерированной на предыдущих кадрах. Метод способен справиться с вариациями формы, перекрытиями, разделением и слиянием объектов.

1.3. Методы оценивания параметров модели

Задача оценивания параметров модели по изображению состоит в том, чтобы определить положение и ориентацию объекта в системе координат камеры. Предполагается, что модель объекта известна. Рассмотрим основные методы решения данной задачи.

1.3.1. Задача P3P

P3P (англ. perspective-3-point) [9] – важная задача, для которой предложено множество решений (рисунок 1.8). Она состоит в том, что нужно найти позу объекта по трем парам соответствий точек объекта и изображения. Теоретически, т.к.

поза представляется шестью параметрами, то трех точек должно быть достаточно для решения этой задачи.

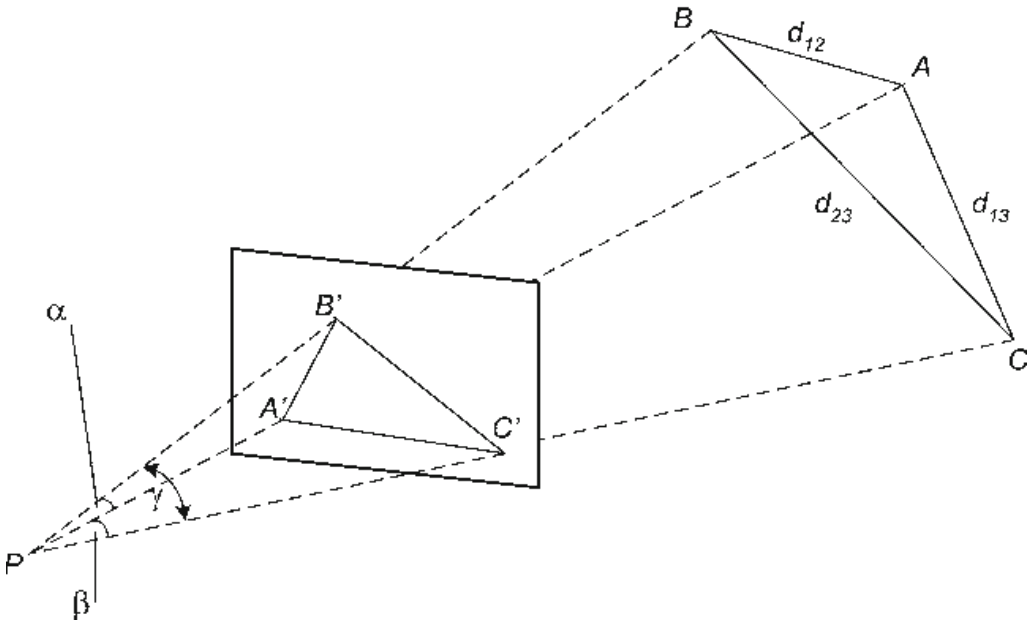


Рисунок 1.8 – Задача P3P

Решение данной задачи основывается на оценке глубины точек, которая может быть получена с помощью решения полиномиального уравнения четвертой степени [10-13]. Вращение может получено в закрытой форме в виде кватерниона [14]. Кроме того, может использоваться сингулярное разложение [13].

Так как уравнение четвертой степени может иметь до четырех действительных корней, то необходима по крайней мере еще одна точка, чтобы разрешить эту неопределенность [10][15].

1.3.2. Задача PnP

Задача PnP – обобщение задачи P3P для большего количества точек. Эта задача –переопределенная (точек значительно больше, чем степеней свободы). Здесь рассматриваются многостадийные оценки координат точек [11], либо виртуальных точек [16] в системе координат камеры. Далее производится трехмерный процесс регистрации [11]. В подходе EPnP [16] трехмерные координаты точек представляются как взвешенная сумма четырех виртуальных контрольных точек. Задача сводится к оценке координат этих виртуальных точек в системе координат камеры. Основное преимущество последнего подхода в снижение количества вычислений (линейное относительно количества точек). Direct Linear Transform (DLT) [15][17] является одношаговым методом. В связи с тем, что система переопределена, это решение чувствительно к выбросам. Альтернативой является решение, предложенное в [18] и [19].

Однако наиболее используемым является решение, основанное на поиске шести неизвестных параметров с помощью минимизации ошибки репроекции методом Levenberg-Marquardt [20].

Выводы по разделу

Была рассмотрена задача оценивания параметров модели по последовательности изображений (видео). Как правило, она делится на два этапа:

1) поиск объекта на изображении с последующим трекингом;

2) непосредственное оценивание параметров модели объекта, используя данные детекции и трекинга.

Первым шагом в процессе трекинга является идентификация интересующих объектов на последовательности кадров и кластеризации пикселей этих объектов. Так как движущиеся объекты как правило являются первичным источником информации, большинство методов фокусируются на детекции таких объектов. Было приведено описание различных методов.

Была рассмотрена задача трекинга – определения траектории движения объекта внутри сцены на изображении. Цель трекинга – определить движение объекта во времени и найти его позицию на каждом кадре последовательности. Трекинг объекта может быть классифицирован на точечный трекинг, методы, основанные на оценке функции ядра и трекинг контуров. Для примера, точечные трекеры выполняют детекцию на каждом кадре, тогда как трекеры, основанные на геометрических свойствах, функции ядра или контурах требуют детекции только когда объект впервые появляется в сцене. Трекинг точек – сложная задача, особенно в случае перекрытия и при ложных детекциях объекта. Распознавание может быть сделано относительно просто с помощью отсечения по порогу при идентификации этих точек.

Также была рассмотрена задача оценивания параметров модели по изображению. Она состоит в том, чтобы определить положение и ориентацию объекта в системе координат камеры. Предполагается, что модель объекта известна. Были рассмотрены основные методы решения данной задачи.

2. МАТЕМАТИЧЕСКАЯ МОДЕЛЬ КАМЕРЫ И ОБЪЕКТА

2.1. RGB-D сенсор, карта глубины

RGB-D сенсор (рисунок 2.1) – периферийное устройство, оснащенное цветной (RGB) камерой, а также инфракрасным датчиком, который позволяет получать *карту глубины*. Сенсор работает в реальном времени с частотой от 30 до 60 кадров в секунду. Буква *D* в аббревиатуре означает *depth*, что переводится с английского как *глубина*.

Рисунок 2.1 – RGB-D сенсор

С помощью RGB камеры сенсор получает цветное изображение. Карта глубины (рисунок 2.2) – числовая матрица, размеры которой совпадают с размерами кадра, в каждой позиции которой расположено число, равное расстоянию от плоскости объектива сенсора до точки пространства, соответствующей пикселю в данной позиции изображения. Таким образом, карта глубины дает информацию о рельефе поверхностей объектов сцены, наблюдаемых из точки обзора.

Рисунок 2.2 – Карта глубины

2.2. Модель камеры

Рассмотрим модель камеры, с помощью которой можно представить процедуру фотоформирования изображения (рисунок 2.3). С помощью проективного преобразования точка в реальной системе координат переводится в картинную плоскость.

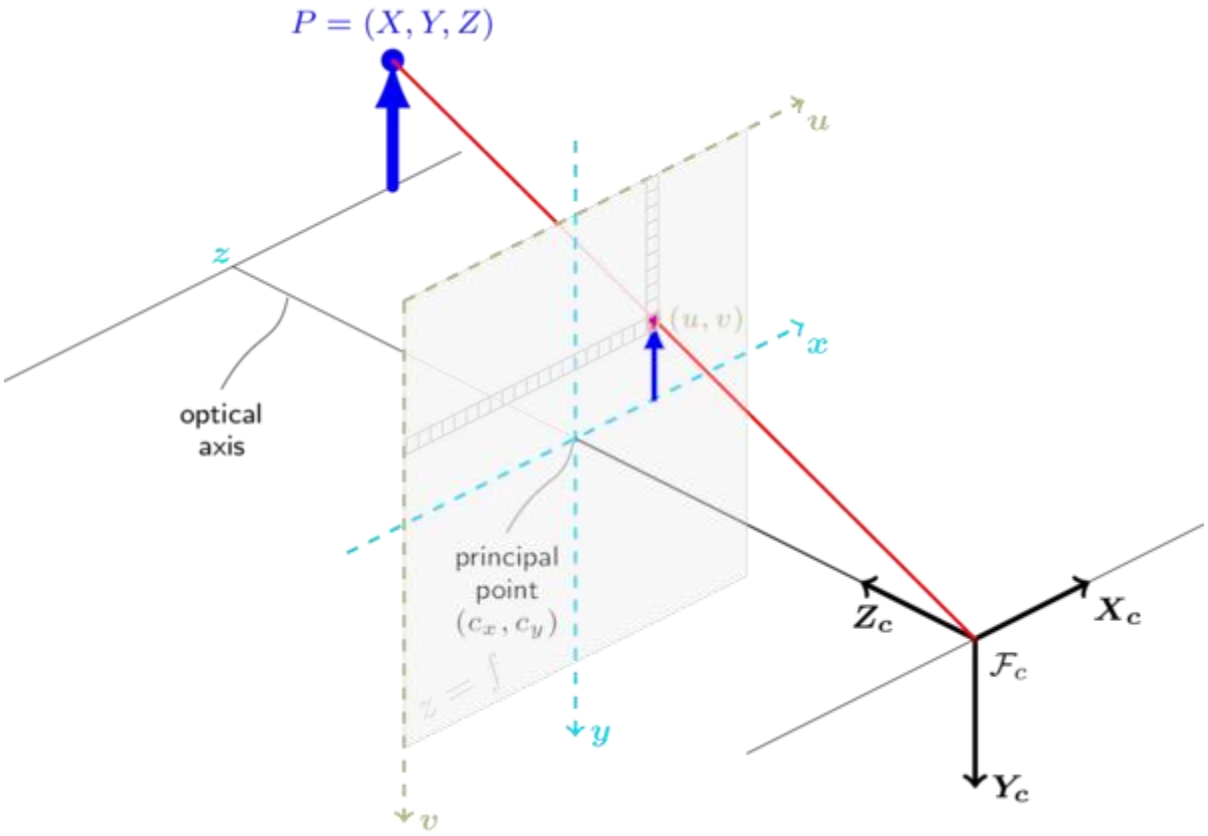


Рисунок 2.3 – Модель камеры

Это преобразование можно записать в виде:

Матрица называется матрицей внутренних параметров камеры.

Коэффициент возникает из-за однородности координат. – фокусное расстояние в горизонтальной и вертикальной длине пикселей. Как правило, они совпадают (это означает, что пиксели – квадратные).

2.2.1. Дисторсия

Радиальная дисторсия – искажение изображения в результате неидеальности параболической формы линзы (рисунок 2.4). Искажения, вызванные радиальной дисторсией, равны нулю в оптическом центре сенсора и возрастают ближе к краям. Как правило, радиальная дисторсия вносит наибольший вклад в искажение изображения.

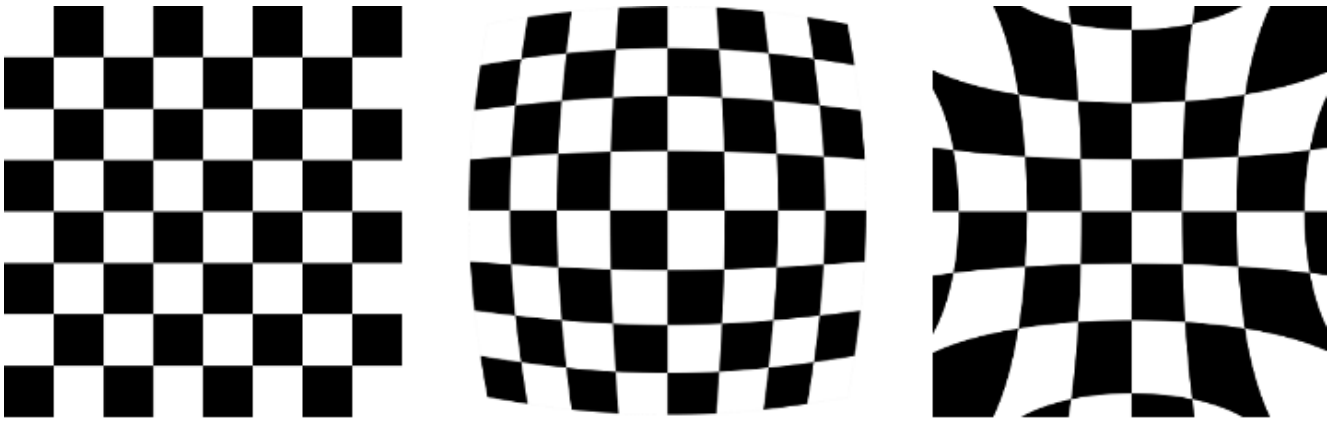


Рисунок 2.4 – Радиальная дисторсия

Тангенциальная дисторсия – искажения изображения, вызванные погрешностями в установки линзы параллельно плоскости изображения (рисунок 2.5).

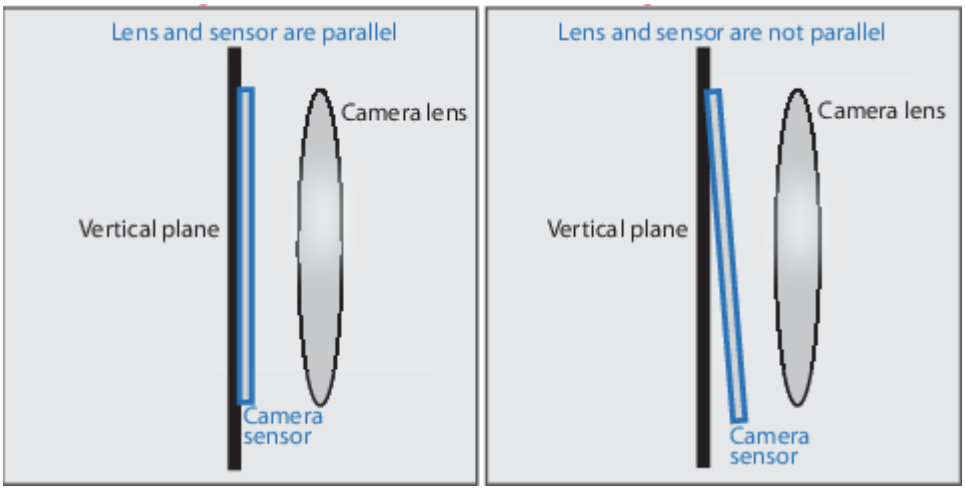


Рисунок 2.5 – Тангенциальная дисторсия

Преобразование дисторсии можно записать с помощью последовательности уравнений:

Здесь k_1, k_2, k_3, k_4, k_5, k_6 – коэффициенты радиальной дисторсии, p_1, p_2 – тангенциальной.

2.2.2. Калибровка камеры

Калибровка камеры – это задача получения внутренних и внешних параметров камеры по имеющимся фотографиям или видео, отснятыми ею. Калибровка камеры часто используется на начальном этапе решения многих задач компьютерного зрения и в особенности дополненной и виртуальной реальности. Кроме того, калибровка камеры помогает исправлять дисторсию на фотографиях и видео.

Матрица внутренней калибровки содержит 5 значимых параметров. Эти параметры соответствуют фокусному расстоянию, углу наклона пикселей и принципиальной точке (точка пересечения плоскости изображения с оптической осью, совпадающая с центром фотографии. В реальных камерах, как правило, бывает немного смещена из-за оптических искажений). Нелинейные параметры внутренней калибровки, такие как коэффициенты дисторсии, также имеют важное значение, хотя и не могут быть включены в линейную модель, описываемую матрицей внутренней калибровки. Большинство современных алгоритмов калибровки камеры определяет их вместе с параметрами линейной части модели [28].

Параметры внутренней калибровки относятся только к камере, но не к сцене, поэтому они изменяются только в том случае, когда меняются соответствующие настройки камеры.

В результате калибровки необходимо найти параметры f_x, f_y, c_x, c_y, а также k_1, k_2, k_3, k_4, k_5, k_6, p_1, p_2.

2.3. Алгоритм калибровки

2.3.1. Поиск паттерна

Калибровка производится с помощью некоторого количества снимков плоского паттерна известной конфигурации, который можно легко обнаружить на изображении. На рисунке 2.6 изображены калибровочные паттерны: шахматный и круговой.

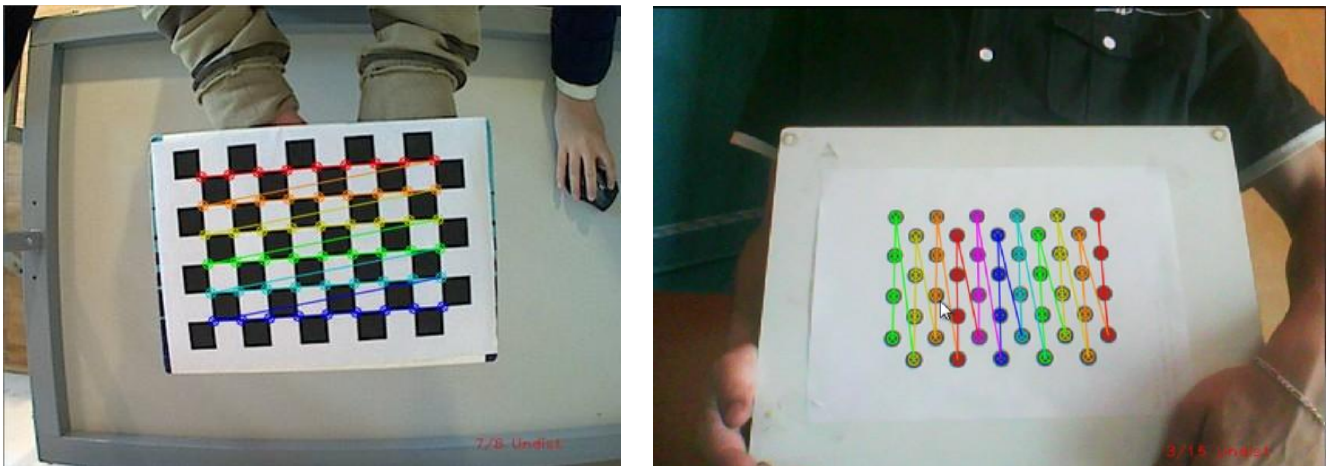


Рисунок 2.6 – Калибровочные паттерны

2.3.2. Детекция размытых изображений

Чтобы точность нахождения паттерна была выше, необходимо обеспечить работу на четких изображениях. Изображения, на которых имеются эффекты размытия в результате движения (рисунок 2.7) не участвуют в калибровке.

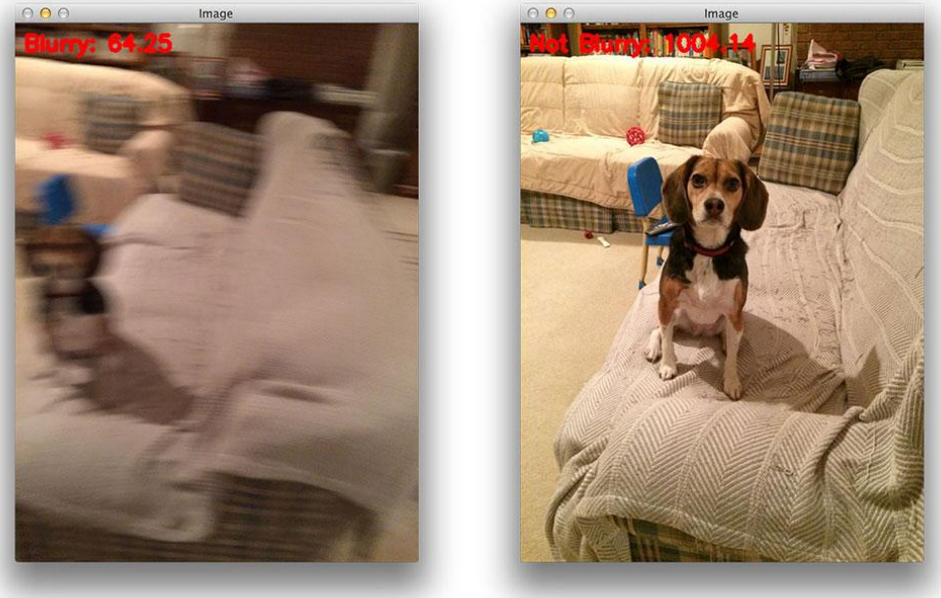


Рисунок 2.7 – Размытое и четкое изображение

Для детекции размытых изображений проведем свертку с фильтром лапласиана (рисунок 2.8) и возьмем дисперсию по пикселям.

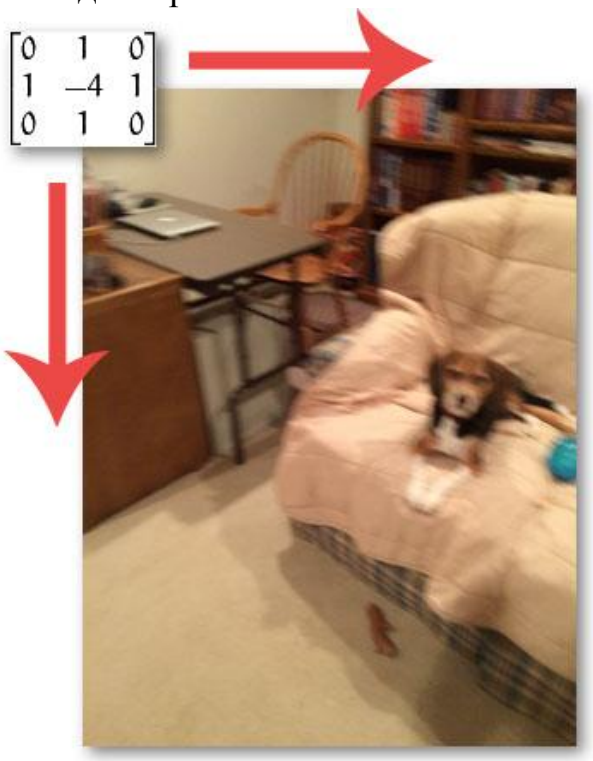


Рисунок 2.8 – Свертка с лапласианом

На размытых изображениях дисперсия будет низкая, на четких – высокая. Поэтому, можно выполнить отсечение по некоторому порогу.

2.3.3. Обеспечение покрытия пространства поз

Для получения более устойчивого результата калибровки необходимо сделать снимки паттерна в разных позах, под разными углами, на разных расстояниях. Для управления этими условиями введем четыре параметра, характеризующие каждый снимок в картинной плоскости.

1) Положение доски по оси X.
2) Положение доски по оси Y.
3) Доля площади доски от всего изображения.
4) Скос доски (отклонения угла от 90 градусов).

Каждый параметр выразим в диапазоне от 0 до 1. Таким образом, каждое изображение представляет собой точку в четырехмерном пространстве. Для калибровки выберем изображения, которые наилучшим образом покрывают все пространство поз (рисунок 2.9). Близкие изображения в выборке также не должны присутствовать, так как они могут вести к смещенности оценки. Для оценки расстояния между изображениями используется метрика L1.

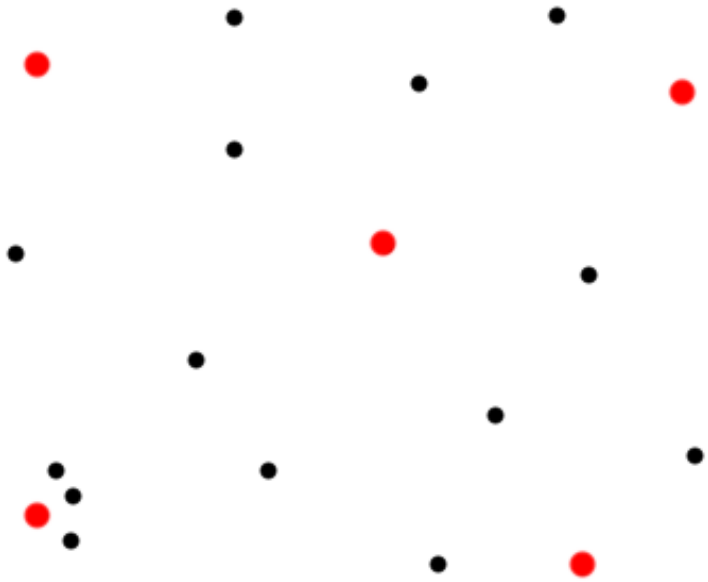


Рисунок 2.9 – Покрытие пространства поз

2.3.4. Устранение дисторсии

После того как камера была откалибрована, можно устранить дисторсию изображений (рисунок 2.10).

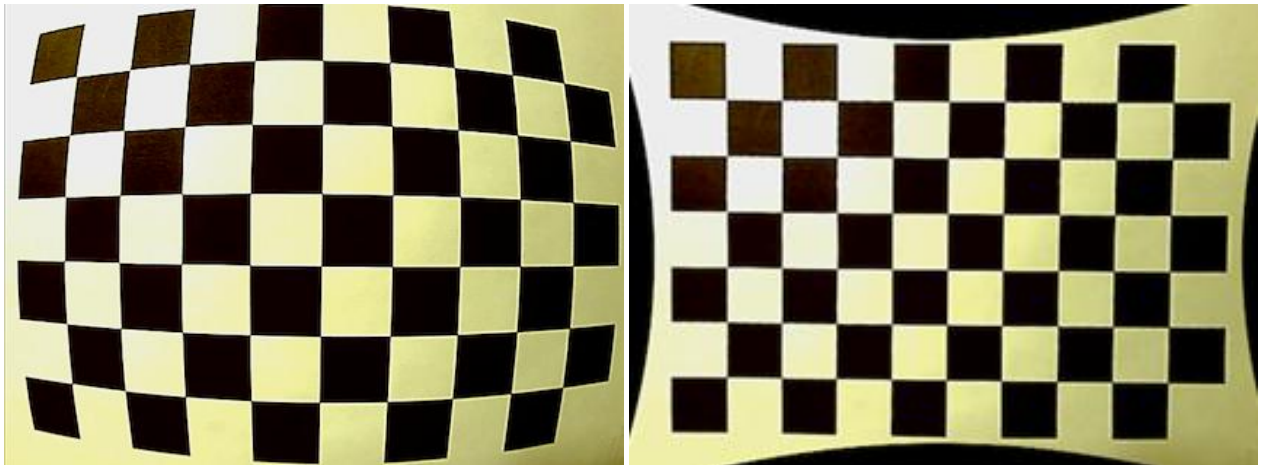


Рисунок 2.10 – Устранение дисторсии

После устранения дисторсии линии на изображении становятся прямыми.

2.4. Модель жесткого объекта

Под жестким понимается объекта, который не деформируется, то есть относительное положение всех его точек не меняется. Зададим объект множеством его точек:

Здесь – координаты точек объекта в системе координат сенсора. Положение объекта в системе координат сенсора можно задать с помощью преобразования вращения и смещения (рисунок 2.11):

Матрица – матрица вращения, – вектор смещения.

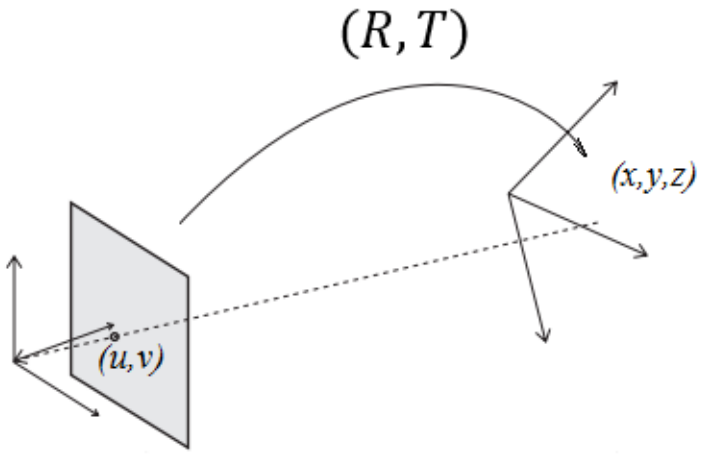


Рисунок 2.11 – Преобразование объекта

Тогда проективные координаты точки объекта можно записать в следующем виде:

Выводы по разделу

В данном разделе описана математическая модель камеры и объекта. Приведены способы перевода точки из реальных координат в проективные. Введено понятие карты глубины.

Также уделено внимание дисторсии – искажению изображения вследствие неидеальности оптических линз.

Кроме того, приведен способ устойчивой калибровки камеры. Для получения более устойчивого результата калибровки необходимо сделать снимки паттерна в разных позах, под разными углами, на разных расстояниях. Для управления этими условиями было введено четыре параметра, характеризующие каждый снимок в картинной плоскости: положение доски, доля площади доски от всего изображения, скос доски (отклонения угла от 90 градусов).

Результатом калибровки является нахождение параметров камеры, а также дисторсии. Приведен метод обеспечения более полного покрытия пространства поз для получения более устойчивого результата калибровки.

Также введено понятие жесткого объекта. Под жестким понимается объект, который не деформируется, то есть относительное положение всех его точек не меняется. Объект был задан множеством его точек. Положение объекта в системе координат сенсора можно задать с помощью преобразования вращения и смещения. Приведена формула преобразования точек из системы координат объекта в систему координат картинной плоскости (проективную систему координат).

3. ОЦЕНИВАНИЕ ПАРАМЕТРОВ ПО RGB-D ДАННЫМ

3.1. Поиск особых точек

Рассмотрим в качестве модели плоский маркер (рисунок 3.1). Такие маркеры часто прикрепляются на объект, и используются для определения позы объекта. Данные маркеры просты в изготовлении (можно распечатать на обычном принтере), при этом они обладают структурой, которую можно выделить на RGB-изображении.

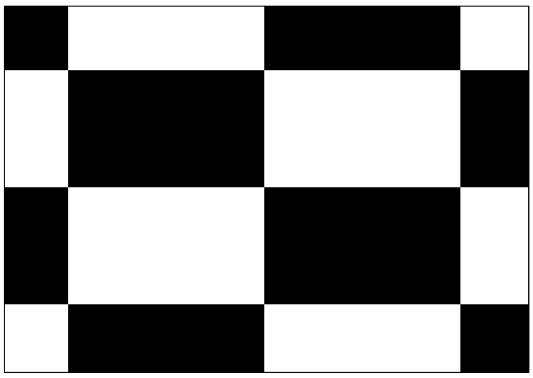


Рисунок 3.1 – Пример маркера

Для быстрого и эффективного определения маркера используется поиск угловых (особых) точек. На рисунке 3.2 можно видеть пример исходного изображения.

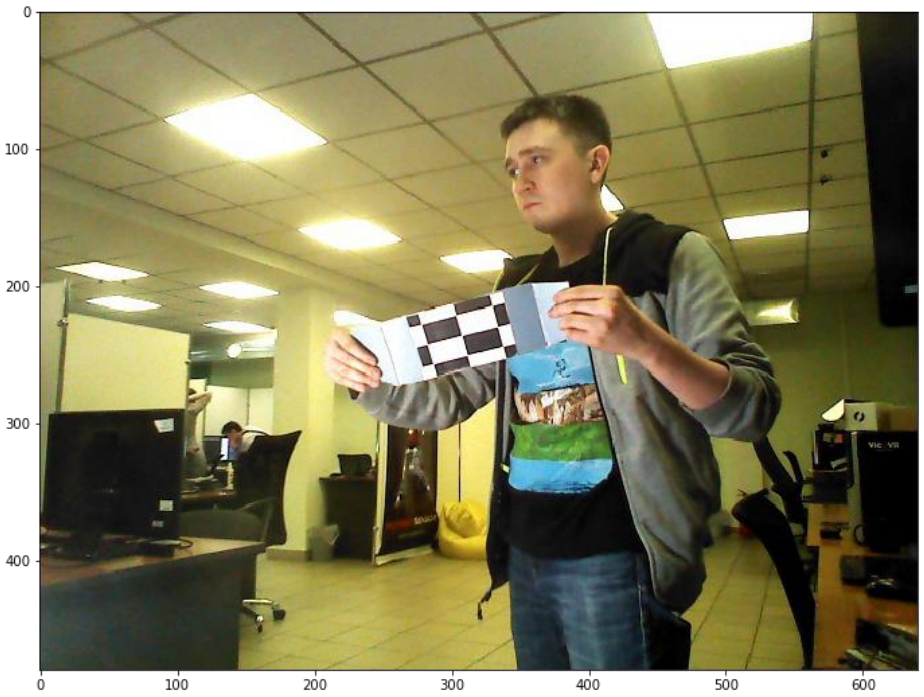


Рисунок 3.2 – Пример входного изображения

Применялись несколько способов детекции особых точек. Они описаны в публикациях [7-9].

Идея поиска особых точек состоит в построении такой функции (фильтра), которая ставит каждому пикселю изображения в соответствие число, которое характеризует его особенность: чем выше это число, тем выше вероятность, что эта точка является особой. В итоге применения этой функции (по сути это дискретная свертка) получается изображение (размер которого совпадает с размером исходного), которое называется откликом фильтра.

В качестве такой функции можно использовать меру изменения окрестности вокруг точки при сдвигах:

Разложив данную функцию в ряд Тейлора, получим матрицу, необходимую для вычисления фильтра:

В качестве фильтра берется следующая величина:

Этот фильтр позволяет найти все точки, окрестность которых значительно меняется при небольших смещения.

Далее рассмотрим частный случай для детекции предложенного маркера: фильтр, который позволит находить шахматные углы.

3.1.1. Поиск шахматных углов.

Шахматные углы обладают особой окрестностью. Во-первых, она центрально-симметрична, независимо от положения, ориентации и перспективных искажений. Кроме того, окрестность вокруг шахматного угла разбита на 4 области черного и белого цвета [21]. Выберем равномерно вокруг точки несколько пикселей (рисунок 3.3).

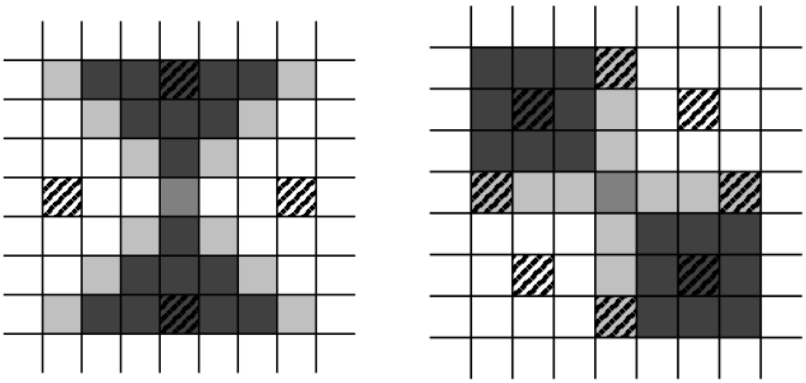


Рисунок 3.3 – 4 и 8 рассматриваемых пикселей

Количество можно варьировать в качестве параметра баланса между точностью и производительностью алгоритма. Рассмотрим наиболее оптимальный вариант – 16 точек (рисунок 3.4).

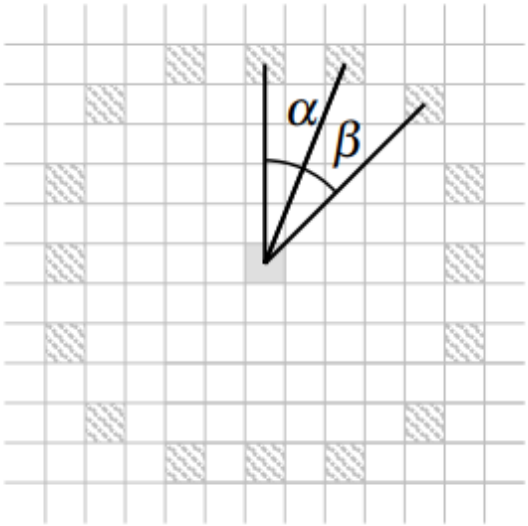


Рисунок 3.4 – 16 рассматриваемых пикселей

При таком выборе точек углы и получаются примерно одинаковыми. Модуль разности между суммами противоположных пикселей под углом 90°. Запишем сумму таких разностей для четырех четверок точек:

Однако, высокий отклик по такому критерию также может давать точка на ребре. Поэтому введем критерий, который будет велик на ребрах, и будем учитывать его с противоположным знаком:

Если взять в качестве критерия разность , высоким откликом также будут обладать точки внутри полос. Поэтому введем еще один критерий – отклонение средней интенсивности в центре и на наружном кольце :

Итоговый отклик фильтра зададим следующим образом:

В общем случае, данный метод может быть обобщен на углы с произвольным количеством линий (рисунок 3.5).

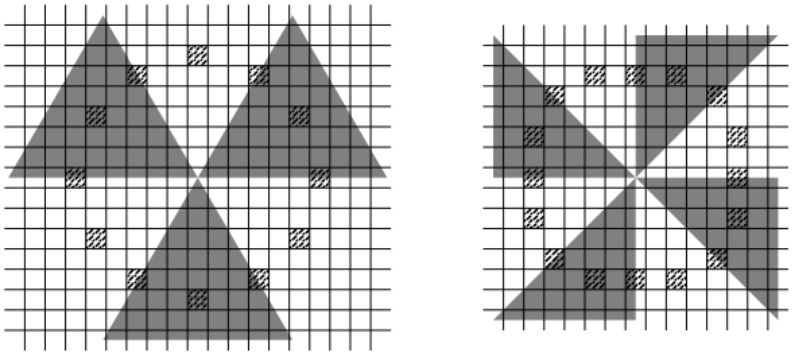


Рисунок 3.5 – Углы различного вида

Маркеры с такими углами могут быть более приемлемыми с точки зрения дизайна, чем с обычными шахматными углами (рисунок 3.6).

Рисунок 3.6 – Пример шлема с маркером с тремя линиями

3.2. Выбор особых точек

Отклик фильтра для исходного изображения (см. рисунок 3.2) приведен на рисунке 3.7.

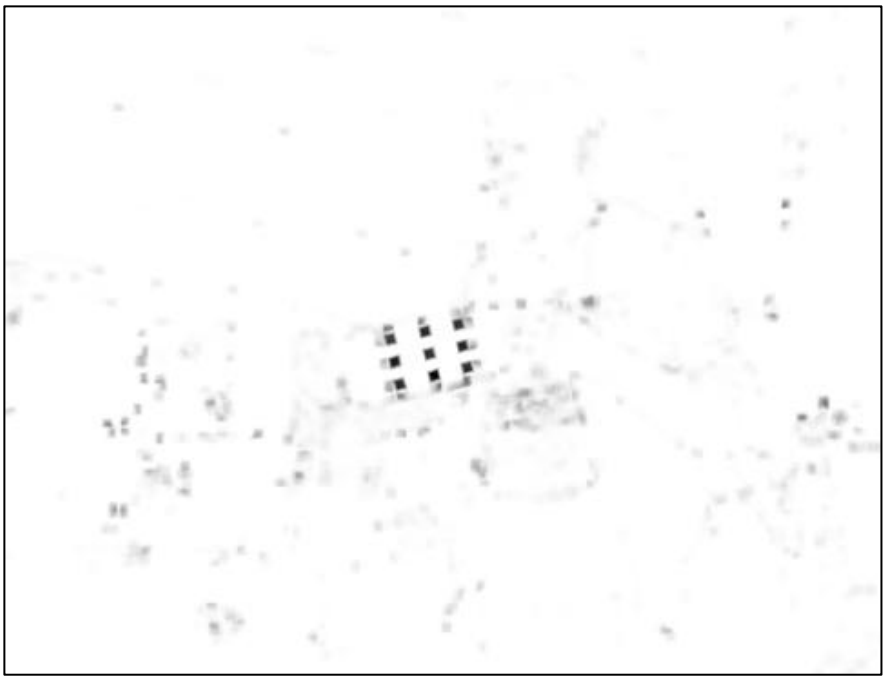


Рисунок 3.7 – Отклик фильтра

С помощью алгоритма Non-maximum suppression (NMS) [22] выберем среди всех пикселей только локальные максимумы (рисунок 3.8).

Рисунок 3.8 – Применение NMS

Далее, можно выполнить отсечение по некоторому порогу, и после этого выбрать определенное количество точек, на которых значения больше, чем у остальных (например, топ 100)

Рисунок 3.4. Особые точки

Из всех найденных особых точек необходимо выбрать те, которые соответствуют углам маркера (рисунок 3.9).

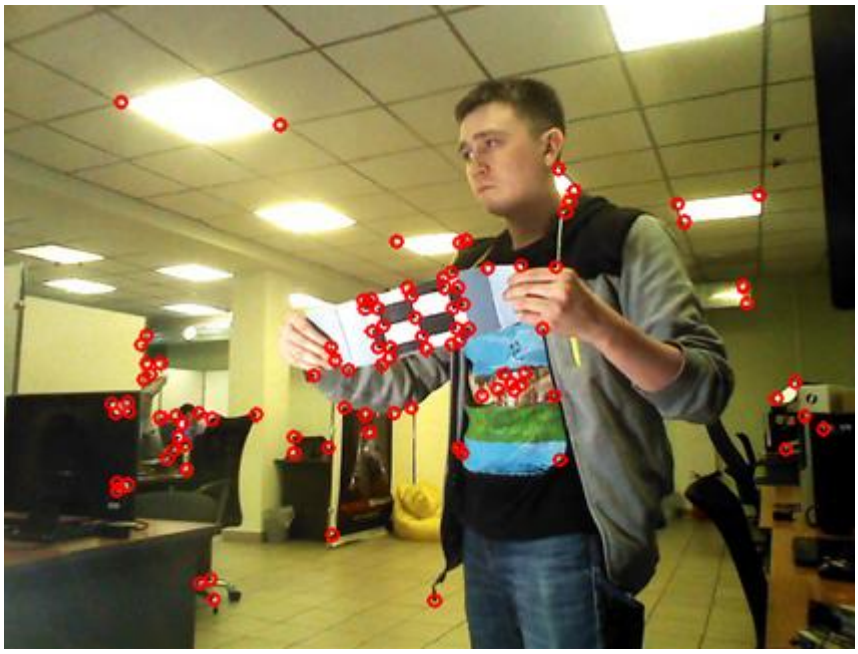


Рисунок 3.9 – Выбранные особые точки

3.3. Поиск подграфа максимального веса

Пусть имеется набор найденных особых точек (рисунок 3.10). Рассмотрим граф *G* с вершинами в этих точках. Так как расстояние между искомыми вершинами маркера ограничено, можно задать для него верхний и нижний порог. Соединим ребрами те вершины, ребра между которыми удовлетворяют обоим порогам.

Искомые ребра маркера характеризуются различным цветом (черный и белый) по разные стороны от ребра. Пусть . Введем вес ребра между этими двумя вершинами с помощью аналога критерия линейного дискриминанта:

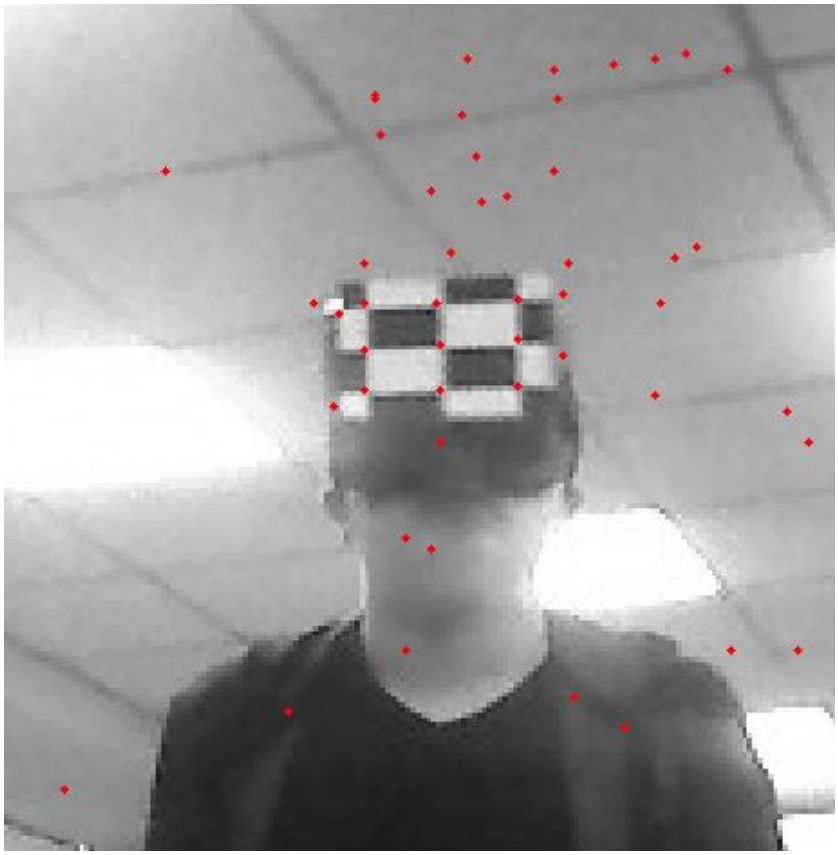


Рисунок 3.10 – Особые точки

Здесь и – математические ожидания и дисперсии величин интенсивностей групп пикселей, выбранных фиксированным образом по одну и по другую сторону ребра соответственно (рисунок 3.11).

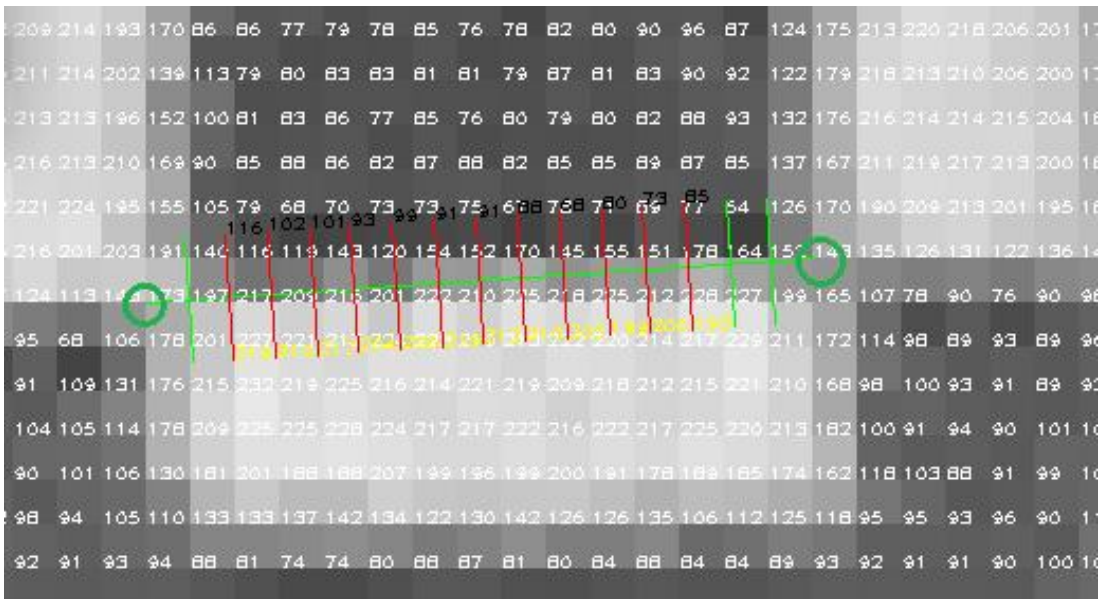


Рисунок 3.11 – Ребро графа

Посчитаем данный вес для всех ребер, которые удовлетворяют условию длины. Получим взвешенный граф (рисунок 3.12). Ребра, вес которых меньше определенного порога можно выкинуть из рассмотрения.

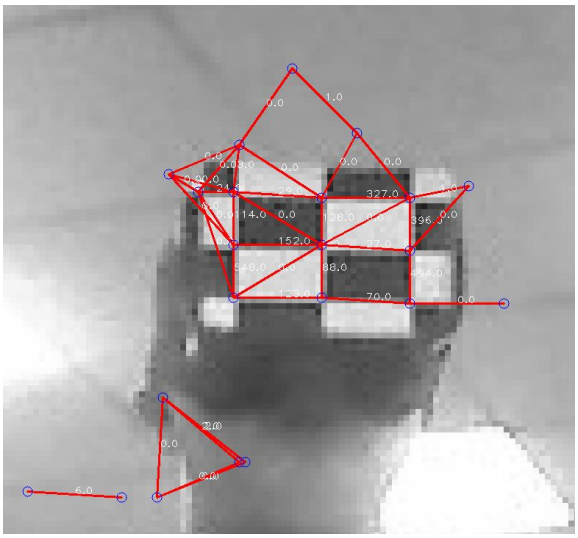


Рисунок 3.12 – Граф особых точек

Из этого графа необходимо выбрать подграф, соответствующий маркеру. Предполагается, что сумма весов его ребер будет максимальной.

В общем случае задача поиска подграфа является *NP*-полной. Поэтому для ее решения используется направленный перебор. Сначала выполняется поиск центральной точки среди вершин, степень которых равна четырем. Среди них выбираются вершины, такие, что ребра, ведущие из них, делятся на пары примерно сонаправленных ребер. Таким образом, выполняется поиск центрального креста

графа. Остальные ребра достраиваются из соображений параллельности. Из всех подграфов выбирается подграф с максимальным весом (рисунок 3.13).

Рисунок 3.13 – Особые точки

3.4. Субпиксельное уточнение позиций

После нахождения особых точек, необходимо уточнить их позиции до субпиксельного уровня для получения более точного результата. Субпиксельное положение может быть найдено из того, что каждый вектор из центра *q* в точку *p* в окрестности ортогонален градиенту изображения в точке p (рисунок 3.14).

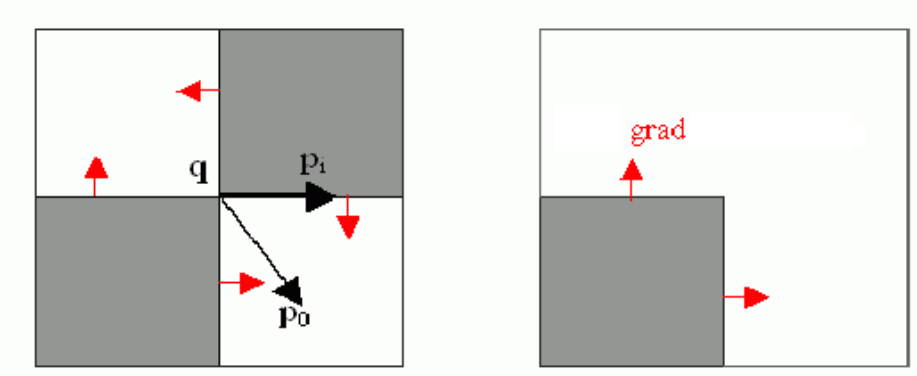


Рисунок 3.14 – Субпиксельное уточнение позиций

Однако, распределения точек, полученные с помощью данного, метода оказываются шумными. Поэтому применялся модифицированный алгоритм Mean-shift.

3.4.1. Алгоритм Mean-Shift

На рисунке 3.15 приведен отклик фильтра в области маркера. Представим его как вероятность особой точки. Тогда этот отклик представляет собой распределе-

ние плотности вероятностей особых точек. Особые точки представляют собой моды этого распределения.

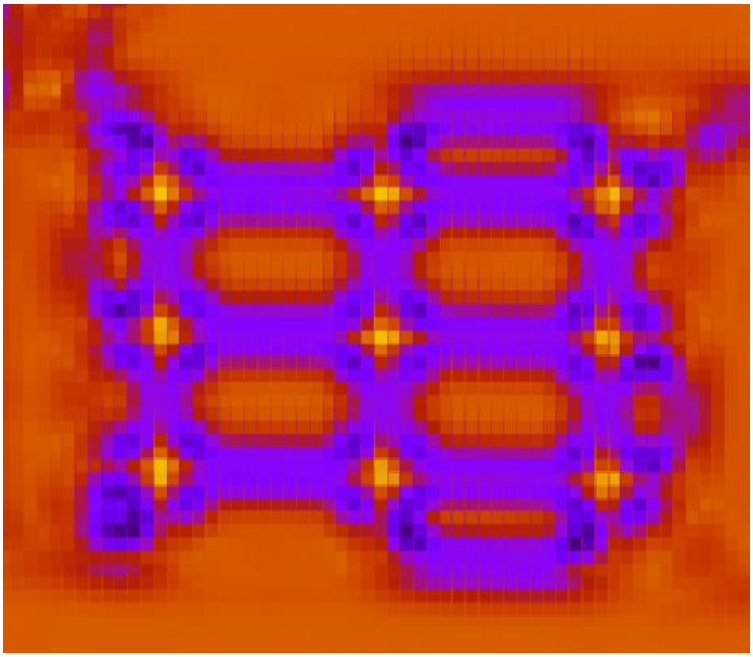


Рисунок 3.15 – Отклик фильтра

Будем искать субпиксельное положение этих точек с помощью итеративного уточнения позиции:

где

сдвиг на текущей итерации;
позиция с пиксельной точностью;
набор точек в окрестности пикселя;
соответствующий отклик фильтра в этих точках.

На рисунке 3.16 приведен пример распределения субпиксельной позиции особой точки в стационарном положении в течение 1000 кадров. Максимальное отклонение от среднего составило 0.0942 пикселя.

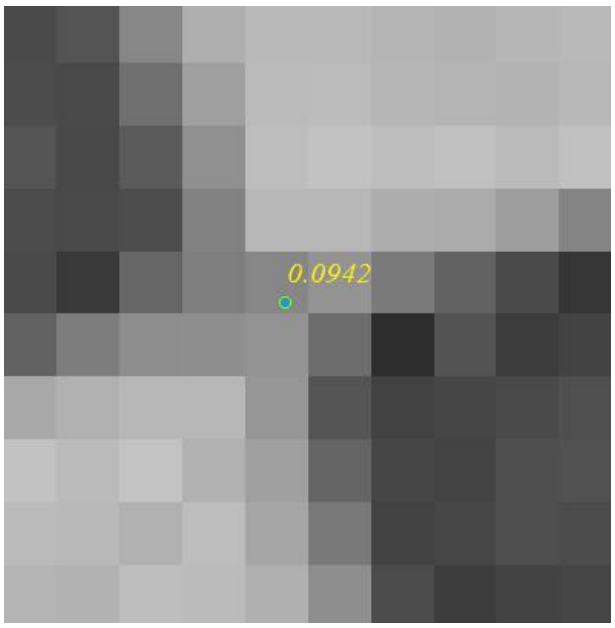


Рисунок 3.16 – Пример субпиксельной оценки

3.5. Дисперсия поиска особых точек

На рисунке 3.17 изображены найденные положения девяти особых точек маркера в стационарном положении в течении некоторого времени. Как видно, разброс найденных точек небольшой, максимальное отклонение от математического ожидания не превышает 0.1 пикселя.

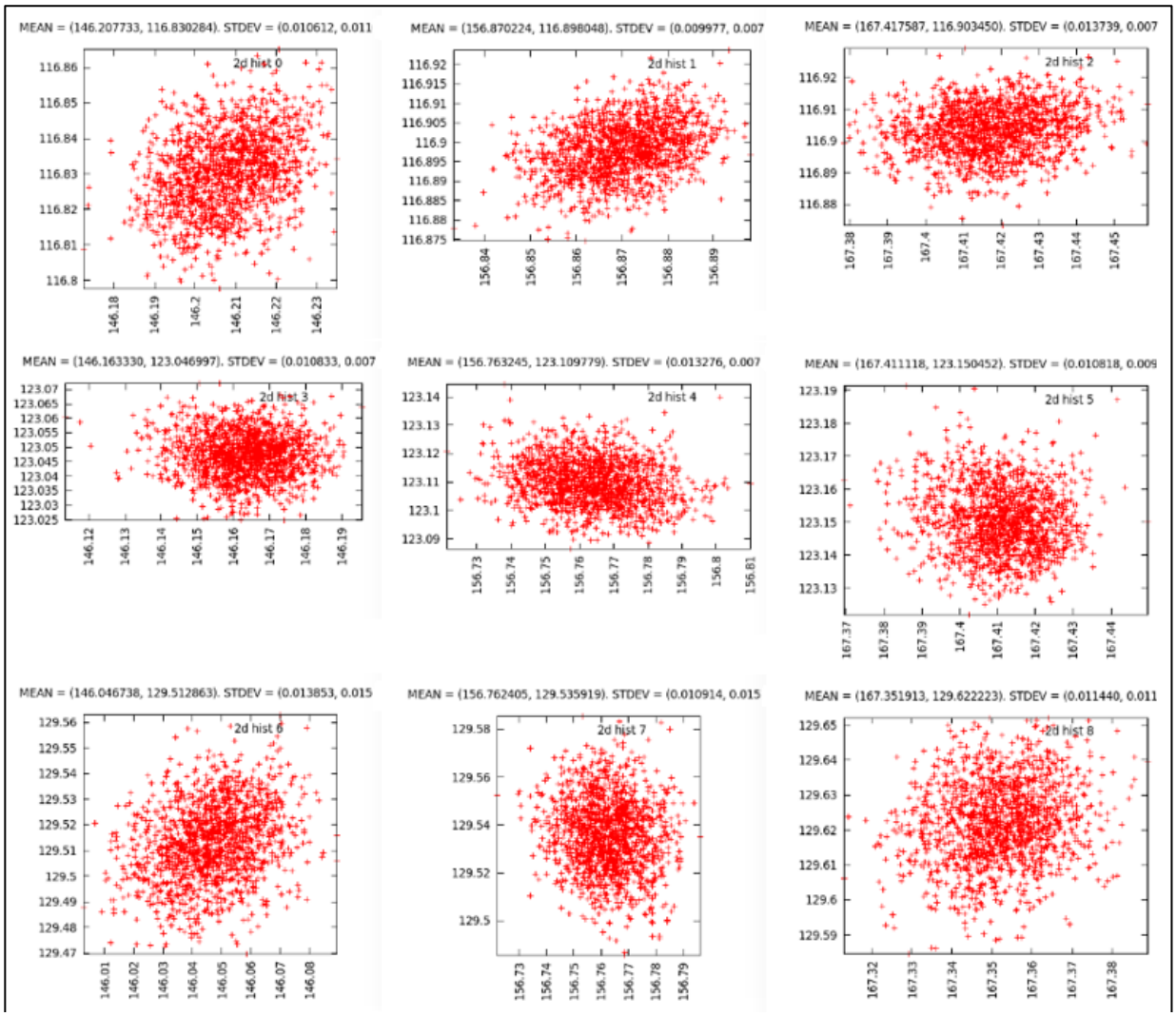


Рисунок 3.17 – Распределение особых точек.

После проведенного эксперимента, в целом, распределение можно считать близким к нормальному.

3.6. Оптимизация ошибки репроекции

Качество соответствия параметров модели и данных задается с помощью ошибки репроекции (рисунок 3.18).

Ошибка репроекции задает суммарное отклонение проекции точек модели при заданном вращении и смещении от проективных координат субпиксельных позиций особых точек:

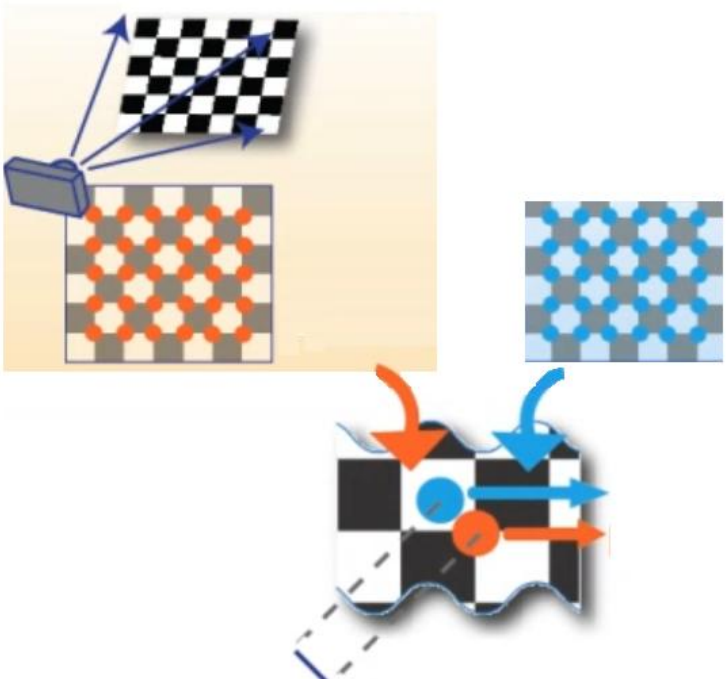


Рисунок 3.18 – Ошибка репроекции

Данная функция оптимизируется с помощью алгоритма Levenberg–Marquardt [20], который на практике показывает хорошие результаты при оптимизации квадратичных функций. Этот метод применяется в задачах PnP [16].

Рассмотрим проблемы, возникшие в ходе оптимизации.

3.7. Двухмодальность оценки

При нефронтальных позах наблюдается двухмодальность оценки (рисунок 3.19).

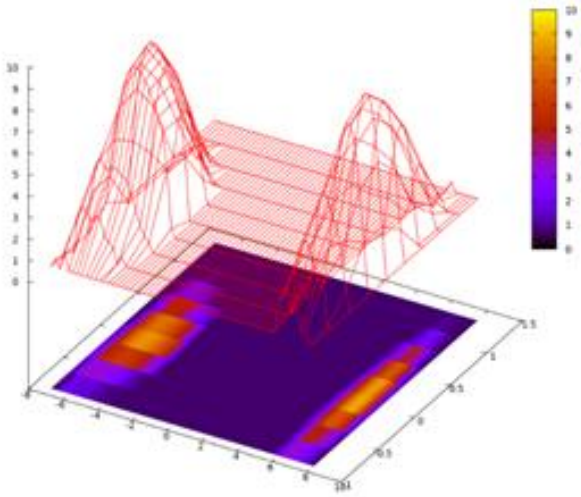


Рисунок 3.19 – Аномалия двухмодальности

Причина этого явления заключается в том, что по RGB-изображению сложно отличить, в какую сторону направлен плоский объект. На рисунке 3.20 приведен пример изображения, по которому сложно понять, куда на самом деле направлен объект: вправо или влево.

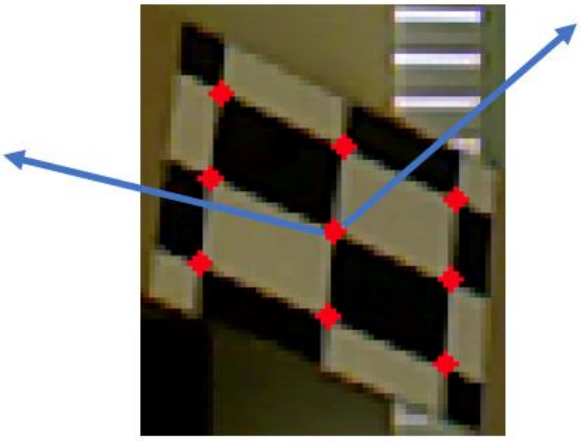


Рисунок 3.20 – Неоднозначность угла

В данном случае, нормаль к маркеру направлена вправо (рисунок 3.21).

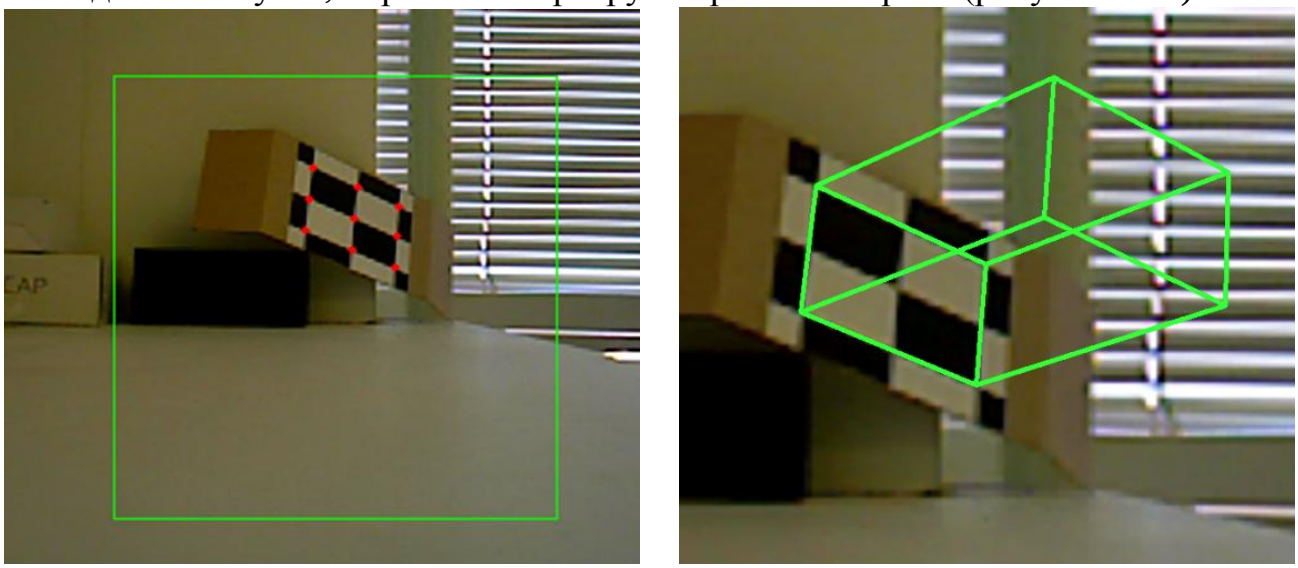


Рисунок 3.21 – Реальное направление маркера

3.8. Аномалия фронтальной позы

При позах, близких к фронтальным, дисперсия оценки угла сильно возрастает (рисунок 3.22).

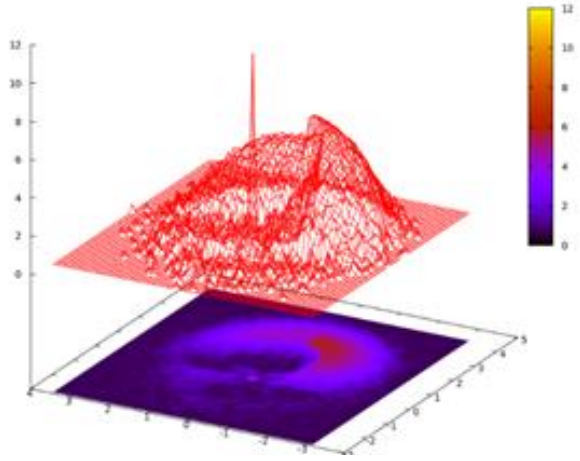


Рисунок 3.22 – Аномалия фронтальной позы

Это объясняется тем, что разброс угла, соответствующего фиксированной ошибке репроекции для фронтальной позы, значительно больше этого же разброса для нефронтальной позы (рисунок 3.23).

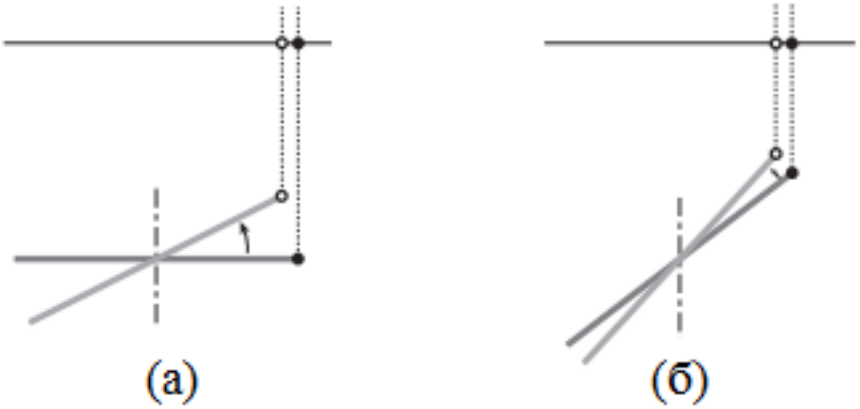


Рисунок 3.23 – Разброса угла: (а) – фронтальная, (б) – нефронтальная

3.9. Карта глубины

Для решения данных проблем была использована карты глубины. Это числовая матрица, размеры которой совпадают с размерами кадра, в каждой позиции которой расположено число, равное расстоянию от плоскости объектива сенсора до точки пространства, соответствующей пикселю в данной позиции изображения.

Рассмотрим следующий кадр с RGB камеры (рисунок 3.24). С помощью нашего алгоритма найдем маркер. Найденные ключевые точки отмечены красными кружками.

Рисунок 3.24 – Применение карты глубины

Данному кадру соответствует карта глубины, которая была заснята сенсором в тот же момент времени. На этой карты выберем точки, которые соответствуют внутренним точкам маркера на цветном изображении (рисунок 3.25). Здесь мы считаем, что цветное изображение и карта глубины отрегистрированы, то есть пиксельные координаты реальных точек на них совпадают.

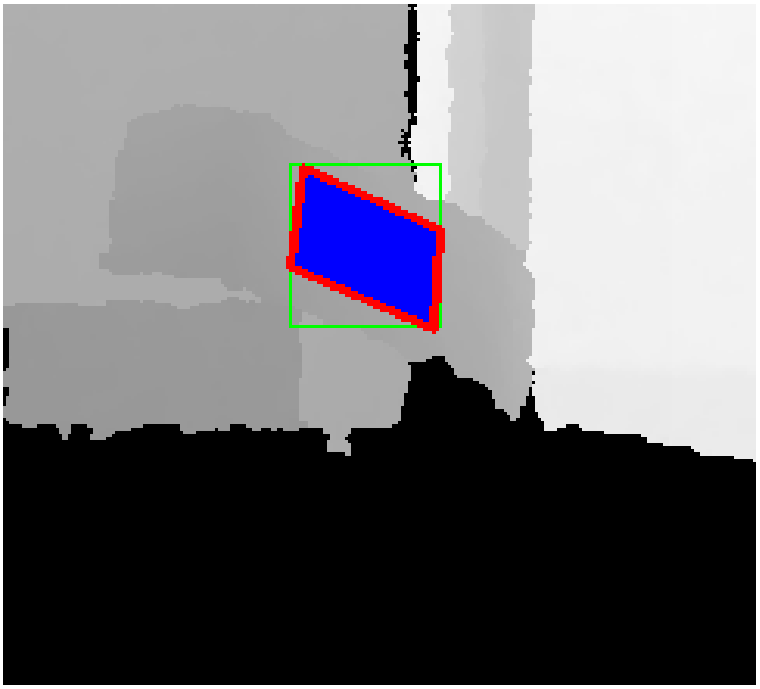


Рисунок 3.25 – Область маркера на карте глубины

Для данной области построим облако точек. С помощью метода главных компонент можно оценить нормаль к плоскости маркера, а также центр масс (рисунок 3.26).

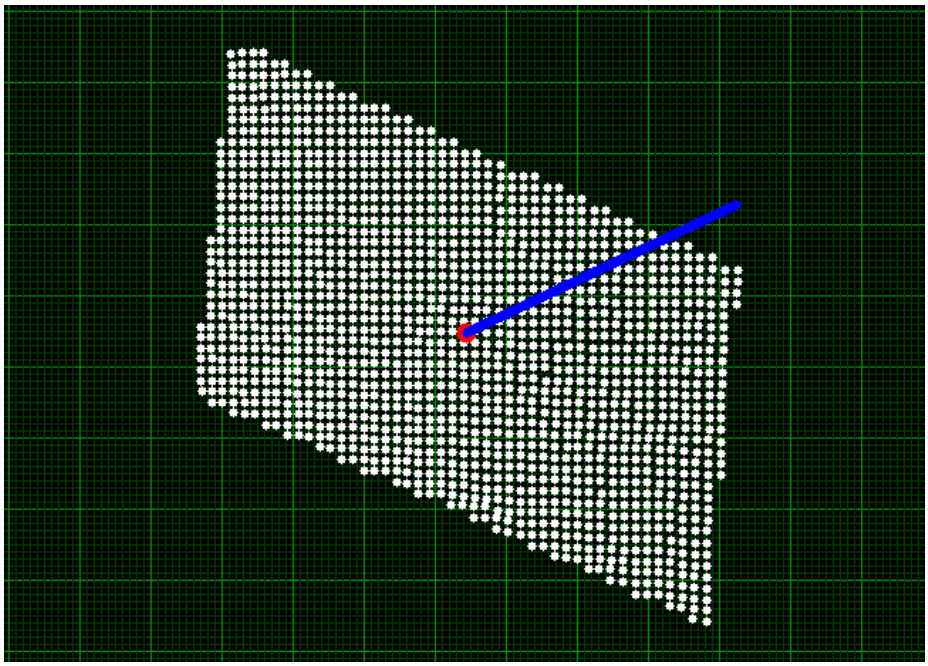


Рисунок 3.26 – Облако точек и нормаль к плоскости маркера

Таким образом, можно устранить неоднозначность (рисунок 3.27).

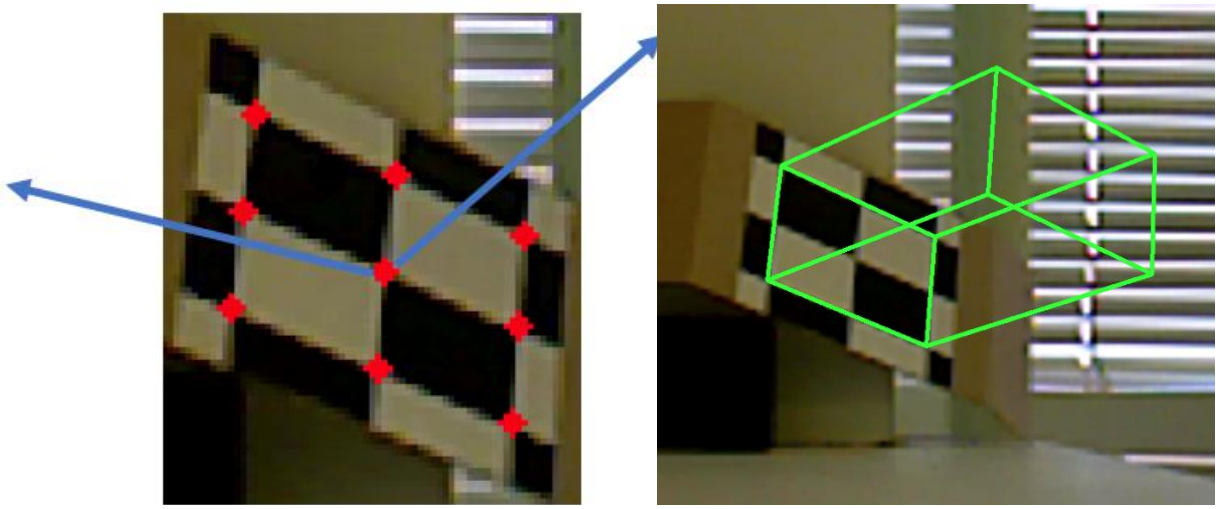


Рисунок 3.27 – Устранение неоднозначности

Кроме того, оценка нормали позволяет значительно снизить дисперсию оценки во фронтальных положениях. На рисунке 3.28 представлено распределение оценки положения центра маркера (по трем координатам) в статичном положении (в миллиметрах).

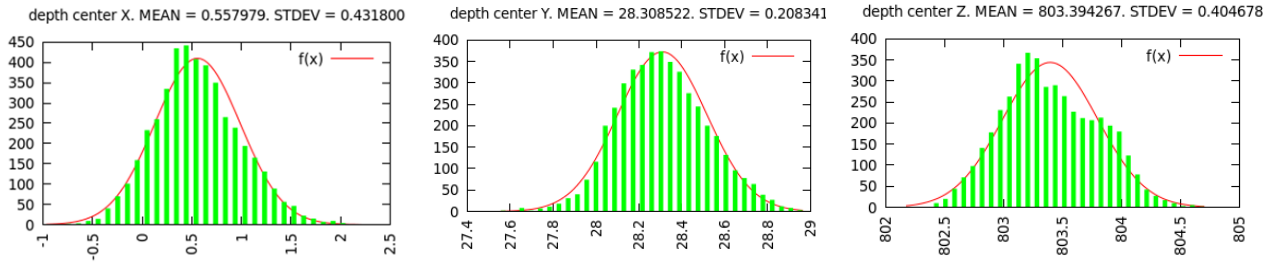


Рисунок 3.28 – Распределение центра маркера

Как видно, дисперсия оценки приемлемо низкая. На рисунке 3.29 также приведено распределение оценки нормали.

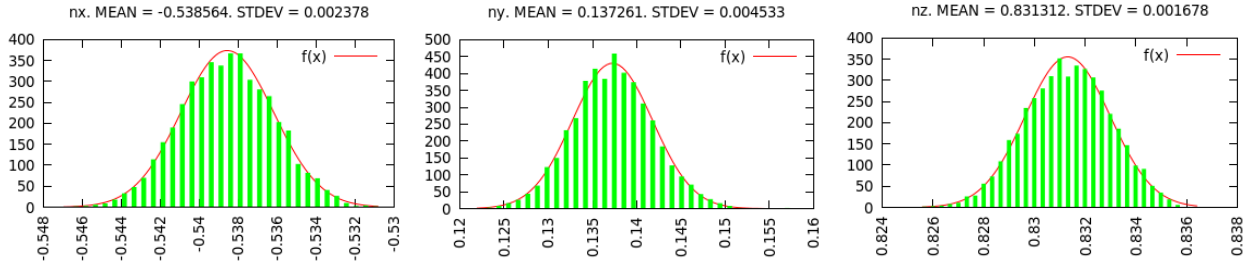


Рисунок 3.29 – Распределение вектора нормали

Оценка с карты глубины может учитываться как после получения результата оценки по RGB изображению, так и в процессе оптимизации. В этом случае к ошибке репроекции также добавляется критерий соответствия карте глубины: норма отклонения от найденного по карте глубины вектора. Критерии необходимо взвесить в зависимости от дисперсий оценок.

Выводы по разделу

В данном разделе была рассмотрена модель плоского маркера. Такие маркеры часто прикрепляются на объект, и используются для определения позы объекта. Данные маркеры просты в изготовлении (можно распечатать на обычном принтере), при этом они обладают структурой, которую можно выделить на RGB-изображении.

Для быстрого и эффективного определения маркера используется поиск угловых (особых) точек. Из всех найденных особых точек необходимо выбрать те, которые соответствуют углам маркера.

Для этого строится граф, веса которого задаются исходя из критерия отличия интенсивностей по разные стороны от ребра. Из этого графа необходимо выбрать подграф, соответствующий маркеру. Предполагается, что сумма весов его ребер будет максимальной. В общем случае задача поиска подграфа является *NP*-полной. Поэтому для ее решения используется направленный перебор. Сначала выполняется поиск центральной точки среди вершин, степень которых равна четырем. Среди них выбираются вершины, такие, что ребра, ведущие из них, делятся на пары примерно сонаправленных ребер. Таким образом, выполняется поиск

центрального креста графа. Остальные ребра достраиваются из соображений параллельности. Из всех подграфов выбирается подграф с максимальным весом.

После нахождения особых точек, необходимо уточнить их позиции до субпиксельного уровня для получения более точного результата.

Качество соответствия параметров модели и данных задается с помощью ошибки репроекции. Ошибка репроекции задает суммарное отклонение проекции точек модели при заданном вращении и смещении от проективных координат субпиксельных позиций особых точек. Данная функция оптимизируется с помощью алгоритма Levenberg–Marquardt, который на практике показывает хорошие результаты при оптимизации квадратичных функций.

Также в разделе описаны проблемы, возникшие в результате оптимизации. Первая проблема – двухмодальность оценки при нефронтальных позах. Вторая – высокая дисперсия оценки при позах, близких к фронтальным. Описано решение данных проблем с помощью использования данных с карты глубины.

4. ИНЕРЦИОННЫЕ ДАТЧИКИ (IMU)

В общем случае объект может не всегда наблюдаем камерой. Его могут перекрывать другие объекты. При значительных перекрытиях продолжать трекинг по RGB-D данным не представляется возможным.

Для решения этой проблемы могут быть использованы инерционные датчики (IMU – inertial measurement units). Также они могут помочь в решении проблемы во фронтальных позах, а также проблемы двухмодальности (рисунок 4.1).

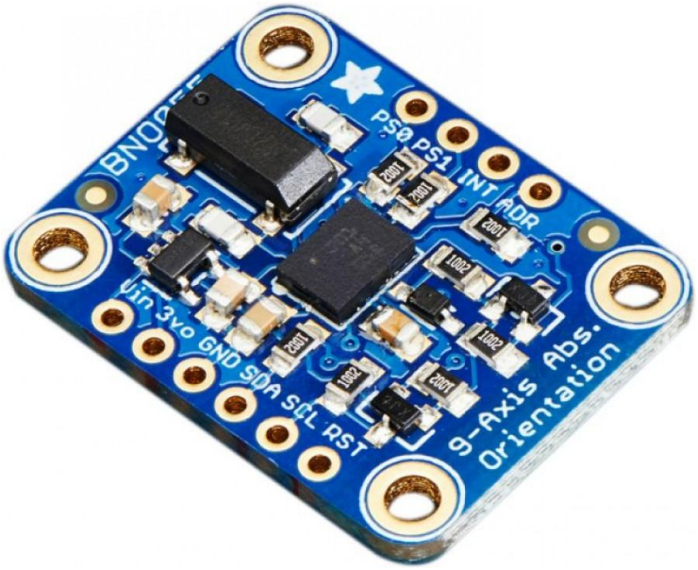


Рисунок 4.1 – Микросхема IMU-датчика

IMU – электронные устройства, которые позволяют измерять некоторые физические характеристики объекта, такие как действующие силы, угловые скорости, магнитное поле земли. Наиболее распространенные датчики – акселерометр, гироскоп и магнитометр. Рассмотрим подробнее каждый из них.

4.1. Акселерометр

Акселерометр (рисунок 4.2) позволяет изменять ускорения тела по трем осям. Данные с акселерометра можно представить в виде:

где – компоненты ускорения по соответствующим осям в момент времени *t*.

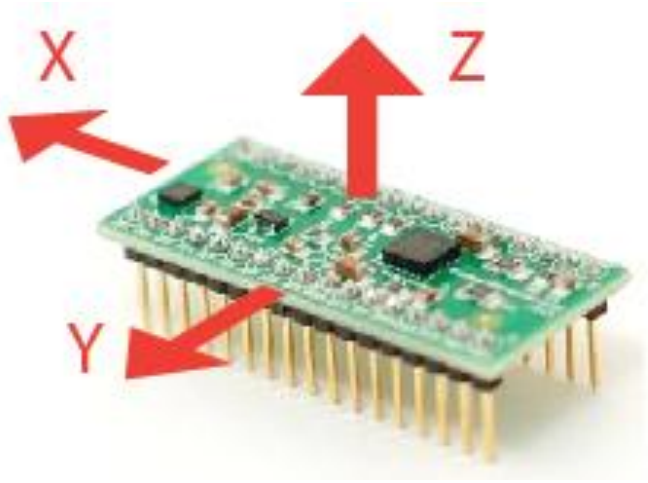


Рисунок 4.2 – Акселерометр

Заметим, что на телефон в состоянии покоя действует сила гравитации, поэтому сырые данные с акселерометра поступают без ее учета. Для отделения влияния гравитации используется различного рода фильтрация. Таким образом, из показаний акселерометра можно получить вектор ускорения и вектор гравитации.

Рабочая частота для наиболее распространенных акселерометров составляет порядка 100-1000 Гц.

4.2. Интегрирование акселерометра

Используя ускорения объекта, можно вычислить его скорость, а используя скорость – вычислить траекторию движения. Движение объекта можно записать в виде дифференциального уравнения:

$$\frac{\quad}{\quad}$$

где – траектория движения, – ускорения. Дважды проинтегрировав, можно найти траекторию (при известных начальных условиях).

Однако, такой подход применим только для очень коротких промежутков времени. Измерения с сенсора имеют некоторую погрешность, которая при двойном интегрировании возрастает квадратично. На рисунке 4.3 приведены сгенерированные данные (200 измерений) для модельного объекта, находящегося в состоянии покоя в течение 1 секунды (200 Гц). Погрешность измерений акселерометра представим как стандартную нормальную случайную величину с дисперсией 1 мм/с^2.

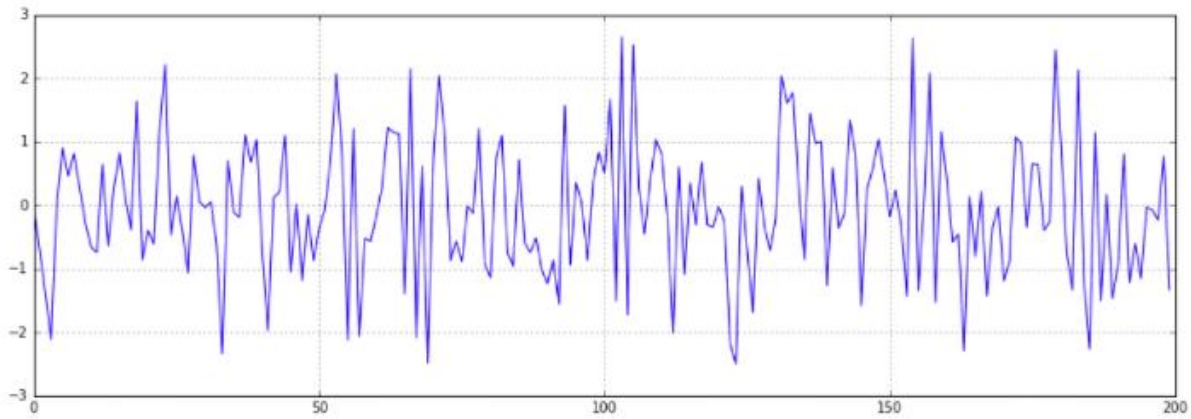


Рисунок 4.3 – Смоделированные ускорения

Проинтегрируем один раз, чтобы найти скорость (рисунок 4.4). Можно видеть, что за 1 секунду ошибка составила примерно 12 мм/с.

Рисунок 4.4 – Скорость

Проинтегрируем еще раз, чтобы получить позицию (рисунок 4.5). Как можно видеть по графику, за 1 секунду ошибка в позиции составила около 1000 мм, то есть около 1 метра.

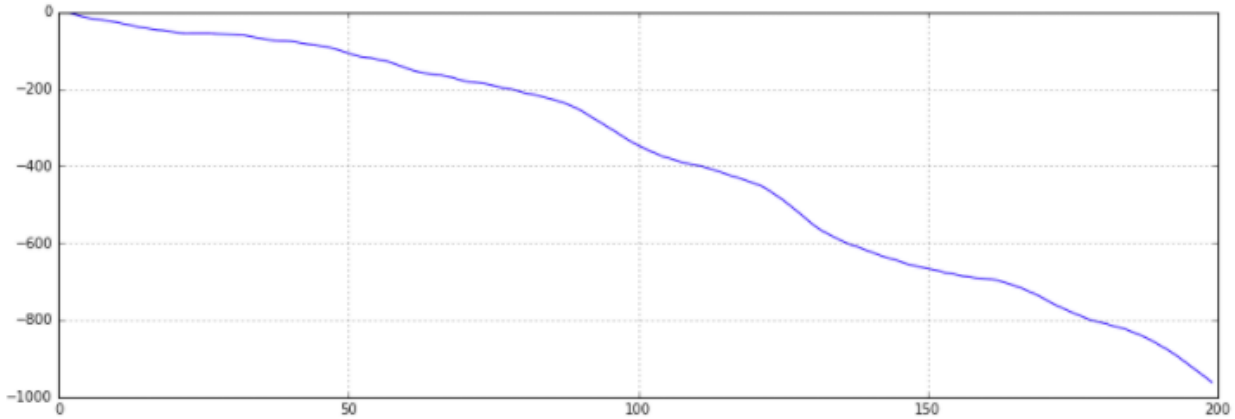


Рисунок 4.5 – Позиция

Очевидно, что получить качественную оценку позиции, используя таким образом данные с акселерометра, не представляется возможным.

4.3. Гироскоп

Гироскоп позволяет измерять угловые скорости вращения тела (рисунок 4.6).

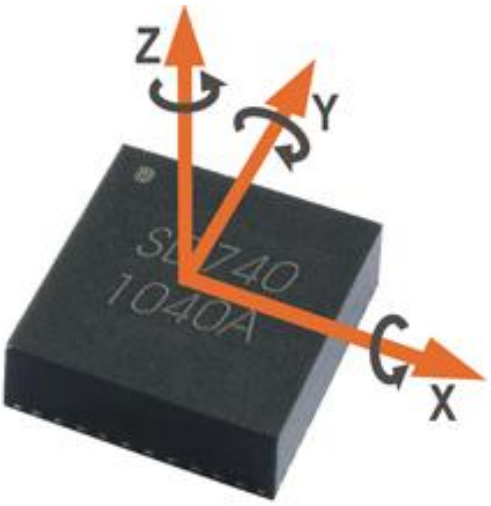


Рисунок 4.6 – Гироскоп

Данные с гироскопа можно представить в виде:

где – угловые скорости вокруг соответствующих осей в момент времени *t*.

Рабочая частота для наиболее распространенных гироскопов составляет порядка 100-1000 Гц.

4.4. Интегрирование гироскопа

Пусть имеется набор измерений угловых скоростей в моменты времени . Найдем изменение вращения объекта между моментами времени t_k и t_{k-1}. Норму угловой скорости можно посчитать по следующей формуле:

где . Нормализуем вектор, после чего проинтегрируем угловую скорость и представим изменение вращения в виде кватерниона:

—

—

—

—

где – угол поворота за прошедший интервал времени, а

— – нормированный вектор угловой скорости.

Так как интегрирование производится один раз, данный способ можно использовать в течение более долгого промежутка времени, в отличие от акселерометра.

4.5. Магнитометр

Магнитометр позволяет получить величину индукции магнитного поля по осям (в теслах). Преимущество магнитометра в том, что его показания не дрейфуют с течением времени. Однако они могут быть очень шумными. На магнитометр могут оказывать воздействие какие-то электронные устройства, металлические конструкции и др.

Магнитометр может частично помочь устранить дрифт других датчиков.

4.6. Применение датчиков

Искомое вращение корректируется с помощью сферической интерполяции кватернионов:

Значение параметра *t* выбирается из практических соображений.

4.7. Трекинг шлема виртуальной реальности

Шлем виртуальной реальности – устройство, создающее эффект визуального присутствия в виртуальном пространстве (рисунок 4.7). Шлем оснащен дисплеем для глаз. Надевается на голову. Шлемы виртуальной реальности применяются во многих областях, таких как проектирование, авиация, медицина, компьютерные игры и др. Шлемы могут использоваться для создания тренажеров и симуляций, например, для тренировки водителей, пилотов, солдат, хирургов и др.

Рисунок 4.7 – Шлем виртуальной реальности

В качестве дисплея в некоторых шлемах может выступать мобильный телефон, например, Samsung GearVR (рисунок 4.8) или Google Daydream (рисунок 4.9).

Рисунок 4.8 – Шлем Samsung GearVR

Рисунок 4.9 – Шлем Google Daydream

4.7.1. Постановка задачи

Для создания наилучшего эффекта присутствия необходимо решить задачу определения положения и ориентации шлема в пространстве.

В рамках исследования в компании 3DiVi данная задача решалась для сенсора VicoVR (рисунок 4.10).

Рисунок 4.10 – Сенсор VicoVR

Это 3D-сенсор, который позволяет получить 3D позиции точек скелета человека в реальном времени (рисунок 4.11).

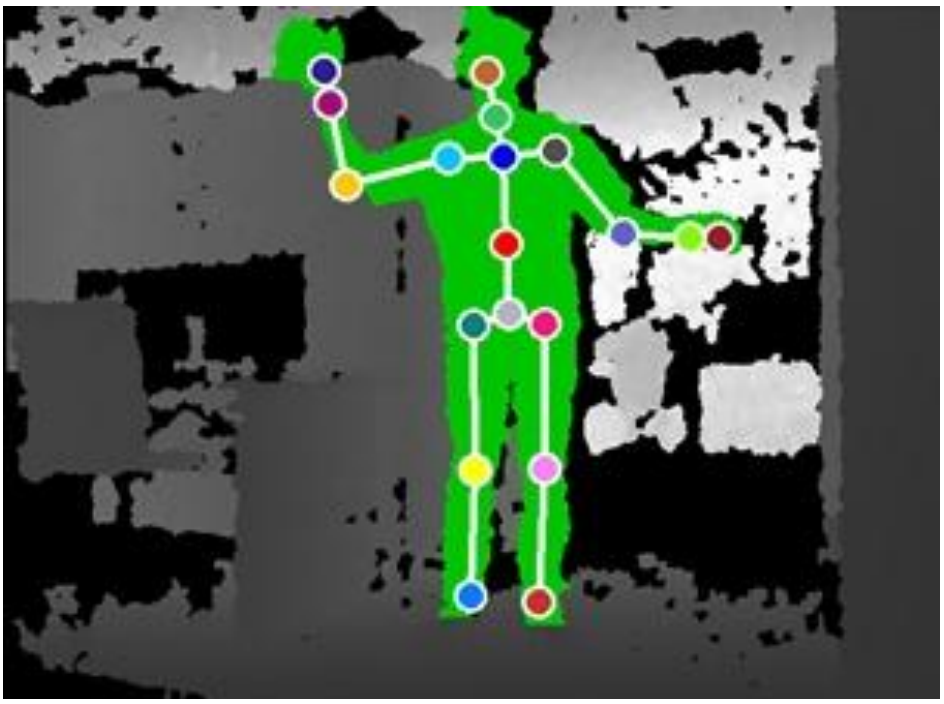


Рисунок 4.11 – Точки скелета

На переднюю поверхность шлема нанесен плоский маркер (рисунок 4.12), который используется для трекинга.

Рисунок 4.12 – Шлем VicoVR

В качестве дисплея в данном шлеме используется смартфон, экран которого разделен по центру на две части – по одной для каждого глаза (для создания эффекта 3D). Телефон оснащен инерционными датчиками.

4.7.2. Задача автокалибровки

Дополнительным фактором для улучшения качества трекинга можно использовать данные о позиции ключевых точек скелета. Эти позиции известны в системе координат RGB-D сенсора (рисунок 4.13).

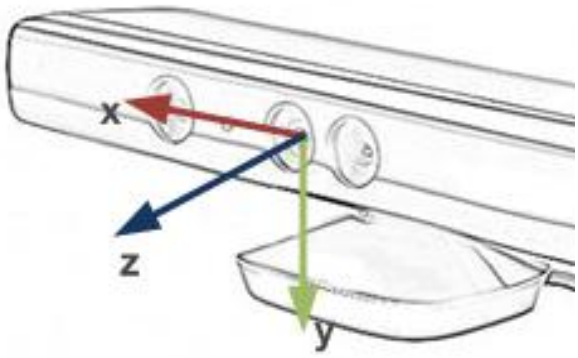


Рисунок 4.13 – Система координат сенсора

Данные с IMU датчиков же поступают в системе координат телефона (рисунок 4.14).

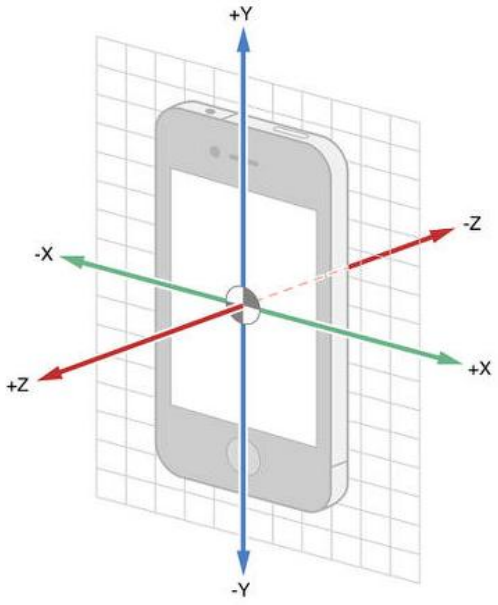


Рисунок 4.14 – Система координат телефона

В связи с этим возникает задача поиска преобразования из одной системы в другую.

В качестве модели расположения центральной точки шлема используется модель head-on-stick (рисунок 4.15).

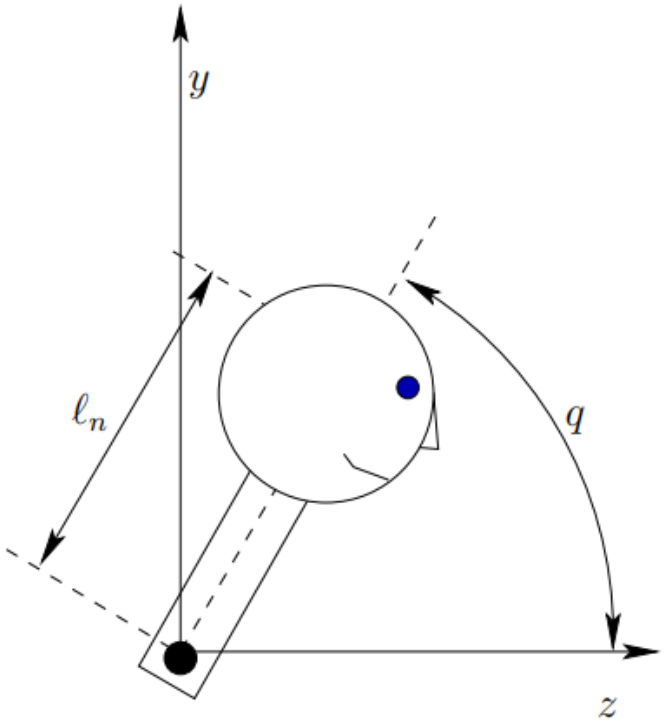


Рисунок 4.15 – Модель head-on-stick.

В рамках этой модели предполагается, что центральная точка шлема двигается по сфере фиксированного радиуса, центр которой находится в шее.

Пусть – трехмерная позиция шеи на кадре *t* в системе координат сенсора, – ускорение с акселерометра, – показания с гироскопа, *l* – вектор от шеи до центральной точки маркера в случае, когда матрица вращения шлема в системе координат сенсора единичная.

Рассмотрим *k* последовательных кадров. Пусть R_i – ориентация шлема в система координат сенсора на *i*-ом кадре. С помощью показателей гироскопа можно оценить матрицы вращения для каждого кадра:

Здесь

– изменение вращения шлема. Примем – единичное вращение.

Используя данные матрицы вращения можно найти положения x_n центральной точки маркера:

Также можно получить значения с помощью интегрирования акселерометра.

Скорость можно найти по формуле:

Здесь – начальная скорость. Здесь мы обозначили

Таким образом получаем:

Заменим :

Перепишем в другом виде:

Для получаем нелинейную систему из *t* уравнений. Неизвестными величинами являются начальное вращение шлема , начальная скорость шлема и вектор *l* модели head-on-stick.

Система может быть решена с помощью общедоступных оптимизационных фреймворков.

Данный подход позволяет корректировать параметры положения и ориентации шлема, когда маркер ненаблюдаем. Однако, система вырождается, когда человек не двигается (в частности, когда человек стоит на месте и его шея неподвижна, но он вращает головой). Кроме того, решение нелинейных систем как правило сводится к итеративному алгоритму оптимизации, что может приводить к существенным вычислительным затратам.

Рассмотрим частный случай задачи, который поможет избежать данных проблем.

4.7.3. Частный случай задачи автокалибровки

В траектории движения человека часто присутствуют относительно прямолинейные участки (с большим радиусом кривизны), а также точки покоя, в которых голова и шея человека не двигаются. Будем искать решение задачи только на этих прямолинейных участках движения, которые заканчиваются состоянием покоя. Это позволяет уйти от неизвестной начальной скорости и принять ее за нулевую. Также будем считать, что RGB-D сенсор расположен горизонтально, то есть ось X его координатной системы параллельна полу (а точнее, перпендикулярна направлению вектора гравитации).

4.7.3.1. Определения прямолинейности участка траектории

Для определения прямолинейности участка воспользуемся данными трехмерных координат шеи, а также показаниями с гироскопа.

Пусть – трехмерная позиция шеи на кадре n, – показания угловой скорости с гироскопа, .

Чтобы оценить прямолинейность траектории применим метод главных компонент (PCA – англ. *principle components analysis*) для трехмерных позиций шеи. Пусть – собственные значения и собственные векторы ковариационной матрицы исходных трехмерных данных, (рисунок 4.16).

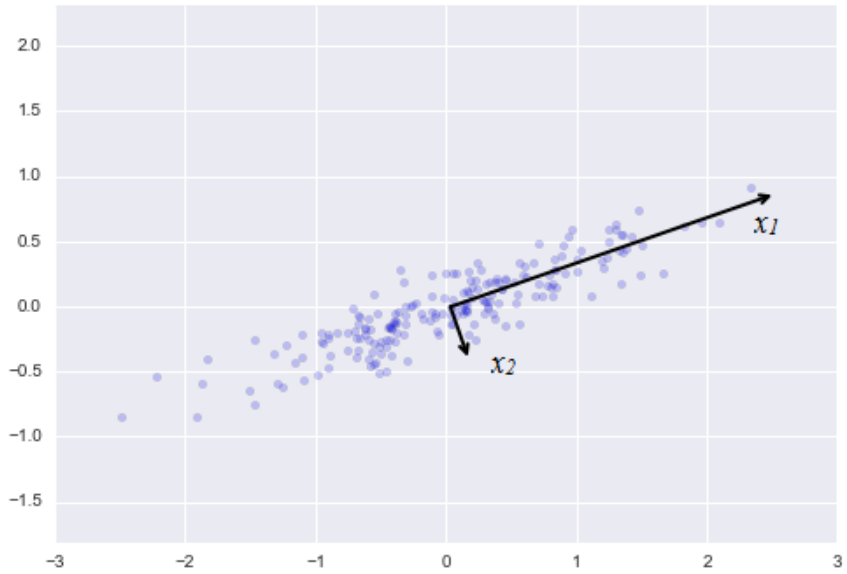


Рисунок 4.16 – Визуализация главных компонент

В качестве критерия рассмотрим величину отношения квадратных корней второго и первого собственного значения:

При идеально прямолинейной траектории данный критерий будет стремиться к нулю. Максимальное же значение может быть равно единице. Будем считать, что траектория является прямолинейной, если выполняется следующее условие:

Здесь – определенный порог, величина которого выбирается на практике. Этот порог задает то, насколько жестко будут отсеиваться траектории, близкие к линейной. В результате практического тестирования было выбрано значение .

Также условием прямолинейности является отсутствие вращения головы на этом участке. Критерием отсутствия вращения можно считать небольшую норму показаний угловой скорости с гироскопа:

При идеально прямолинейной траектории данный критерий будет стремиться к нулю. Будем считать, что вращение приемлемо мало, если выполняется условие:

На практике был выбран порог .

4.7.3.2. Определение вращения на прямолинейных участках

Предположим, что мы нашли участок, удовлетворяющий обоим критериям прямолинейности. Рассмотрим показания инерционных датчиков на этом участке.

Пусть , – вектора ускорения в системе координат телефона на нулевом кадре последовательности. Второй критерий прямолинейности обеспечивает близость матриц к единичным.

Применим метод главных компонент к векторам . Пусть – соответствующие собственные значения и собственные векторы, .

Как дополнительный фактор прямолинейности можно ввести критерий, аналогичный (4.16), и для ускорений .

Заметим, что вектор главных компонент определен с точностью до знака. Для позиций шеи выберем тот, который соответствует направлению движения. Чтобы выбрать знак собственно вектора для ускорений, будем рассматривать только участки, которые заканчиваются остановкой. Ускорения в конце участка будут отрицательными, благодаря чему мы можем выбрать знак собственного вектора.

Таким образом, получаем, что нужно найти такое вращение , которое переводит вектор в вектор наилучшим образом. С помощью акселерометра, можно получить значения вектора гравитации g в системе координат телефона. Также будем использовать то, что ось X сенсора расположена перпендикулярно вектору гравитации.

Будем искать матрицу в виде произведения двух матриц:

где Q – матрица, приводящая вектор в вектор , а – матрица вращения вокруг вектора .

Получим следующую систему матричных уравнений:

Второе уравнение – условие перпендикулярности вектора гравитации (после преобразования) и оси oX (равенство нулю скалярного произведения).

Найдем матрицу Q. Пусть , , . Тогда матрицу Q можно найти по следующей формул:

Здесь – единичная матрица, а – кососимметричная матрица представления векторного произведения:

После применения матрицы Q нужно «докрутить» получившийся вектор гравитации так, чтобы он стал перпендикулярен оси oX. Вращать его нужно вокруг вектора , чтобы он не менялся, сохраняя справедливость первого уравнения.

Матрицу вращения вокруг заданного вектора на заданный угол найдем с помощью формул Родрига. Введем вспомогательную матрицу:

Здесь – нормированные координаты вектора . Тогда вращение вокруг вектора на угол можно найти следующим образом:

После подстановки во второе уравнение системы (4.21) вместо *P*, получим уравнение следующего вида:

где – известные величины.

Действительно, в результате вычислений синус и косинус входят не более, чем в первой степени, при этом уравнение равносильно равенству нулю первой компоненты получившегося вектора.

Уравнение (4.26) легко решается преобразованием в синус суммы. Оно будет иметь на интервале либо два решения, либо будет вырожденным. Это очевидно из построения системы: на плоскости существует два вектора (противоположно направленных), которые перпендикулярны заданному (если он не перпендикулярен этой плоскости). Выбор нужного угла из двух осуществляется из соображений трекинга (выбирается угол, который более согласован с оценкой на предыдущих кадрах).

Таким образом, находим матрицу *P*:

Итоговое вращение находим как произведение матриц *P* и *Q:*

Найденное вращение показывает ориентацию шлема в начале выбранного участка. Однако, на практике необходимо находить вращение в последний момент времени. Поэтому, последовательность рассматривается в обратном порядке (так, как если бы время шло назад), при этом инвертируются знаки всех показателей инерционных датчиков (вектора умножаются на -1).

Выводы по разделу

В данном разделе описано применение инерционных датчиков. В общем случае объект может не всегда наблюдаем камерой. Его могут перекрывать другие объекты. При значительных перекрытиях продолжать трекинг по RGB-D данным не представляется возможным. Для решения этой проблемы могут быть использованы инерционные датчики. Также они могут помочь в решении проблемы во фронтальных позах, а также проблемы двухмодальности. Наиболее распространенные датчики – акселерометр, гироскоп и магнитометр. С помощью гироскопа может быть получено вращение, которое корректируется с помощью сферической интерполяции кватернионов.

Также была рассмотрена задача трекинга шлема виртуальной реальности – устройства, создающего эффект визуального присутствия в виртуальном пространстве. Шлем оснащен дисплеем для глаз. Надевается на голову. Шлемы виртуальной реальности применяются во многих областях, таких как проектирование, авиация, медицина, компьютерные игры и др. Шлемы могут использоваться для создания тренажеров и симуляций, например, для тренировки водителей, пилотов, солдат, хирургов и др. В качестве дисплея в некоторых шлемах может выступать мобильный телефон. Для создания наилучшего эффекта присутствия необходимо решить задачу определения положения и ориентации шлема в пространстве. На переднюю поверхность шлема нанесен плоский маркер, который используется для трекинга.

В качестве дисплея в данном шлеме используется смартфон, экран которого разделен по центру на две части – по одной для каждого глаза (для создания эффекта 3D). Телефон оснащен инерционными датчиками. Дополнительным фактором для улучшения качества трекинга были использованы данные о позиции ключевых точек скелета. Эти позиции известны в системе координат RGB-D сенсора. Данные с IMU датчиков же поступают в системе координат телефона. В связи с этим возникает задача поиска преобразования из одной системы в другую. В рамках этой модели предполагается, что центральная точка шлема двигается по сфере фиксированного радиуса, центр которой находится в шее. Система может быть решена с помощью общедоступных оптимизационных фреймворков. Данный подход позволяет корректировать параметры положения и ориентации шлема, когда маркер ненаблюдаем.

Также был рассмотрен более простой частный случай задачи. В траектории движения человека часто присутствуют относительно прямолинейные участки (с большим радиусом кривизны), а также точки покоя, в которых голова и шея человека не двигаются. Решение задачи искалось только на этих прямолинейных участках движения, которые заканчиваются состоянием покоя. Это позволяет уйти от неизвестной начальной скорости и принять ее за нулевую

Найденное вращение показывает ориентацию шлема в начале выбранного участка. Однако, на практике необходимо находить вращение в последний момент времени. Поэтому, последовательность рассматривается в обратном порядке.

5. АНАЛИЗ РЕЗУЛЬТАТОВ РАБОТЫ

5.1. Дисперсия оценки вращения

На рисунках 5.1-5.3 приведены результаты замеров точности определения углов ориентации маркера в градусах. В названии рисунков приведены математические ожидания и среднеквадратичные отклонения соответствующих величин.

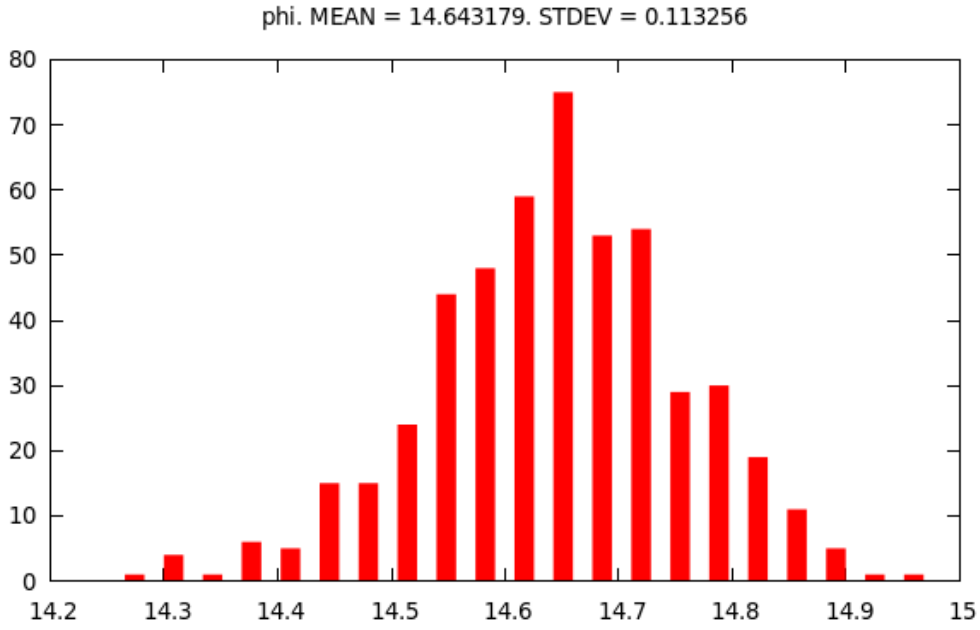


Рисунок 5.1 – Гистограмма распределения угла вращения относительно оси X

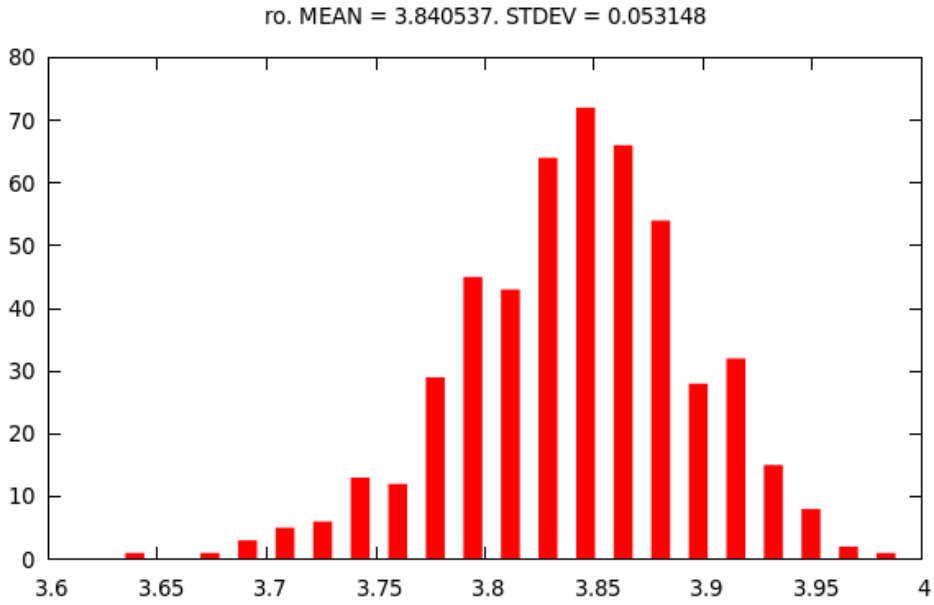


Рисунок 5.2 – Гистограмма распределения угла вращения относительно оси Y

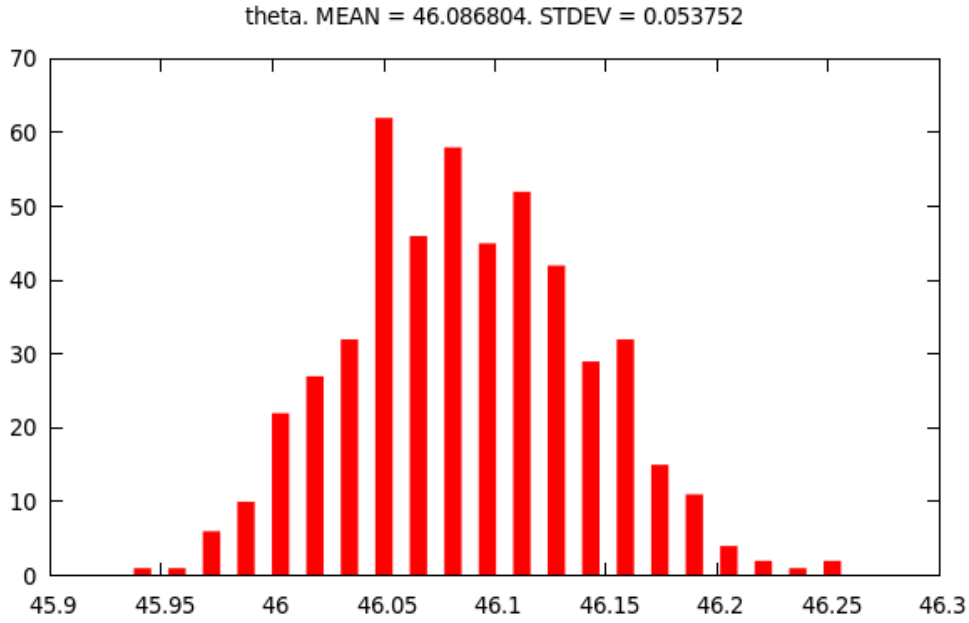


Рисунок 5.3 – Гистограмма распределения угла вращения относительно оси Z

5.2. Дисперсия оценки смещения

В ходе работы был проведен эксперимент по определению параметров смещения. На рисунках 5.4-5.6 приведены результаты определения точности смещения маркера в системе координат сцены в миллиметрах. В названии рисунков приведены математические ожидания и среднеквадратичные отклонения соответствующих величин. Маркер находился на расстоянии около полутора метров.

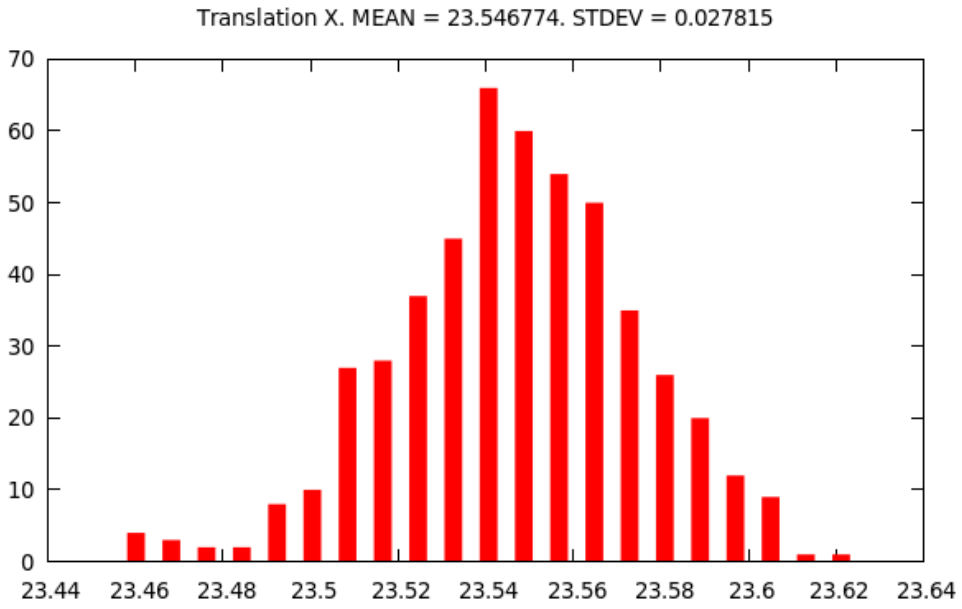


Рисунок 5.4 – Гистограмма распределения смещения по оси X

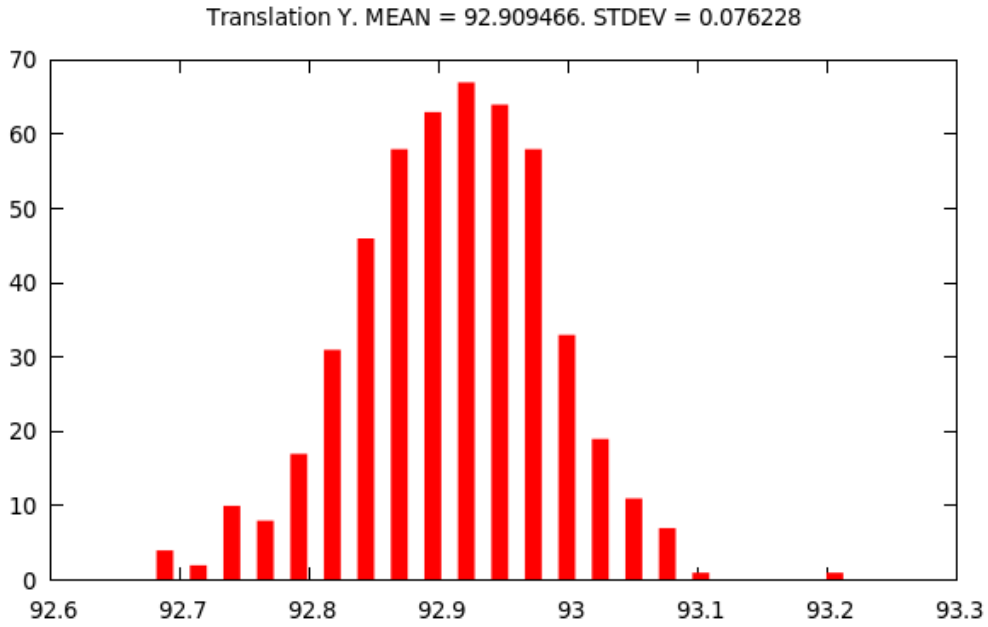


Рисунок 5.5 – Гистограмма распределения смещения по оси Y

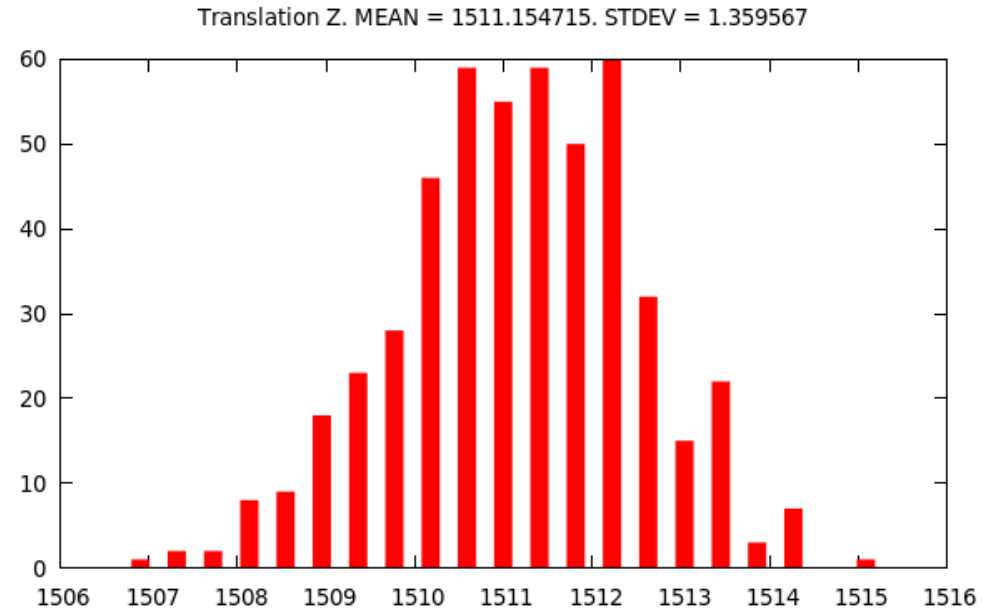
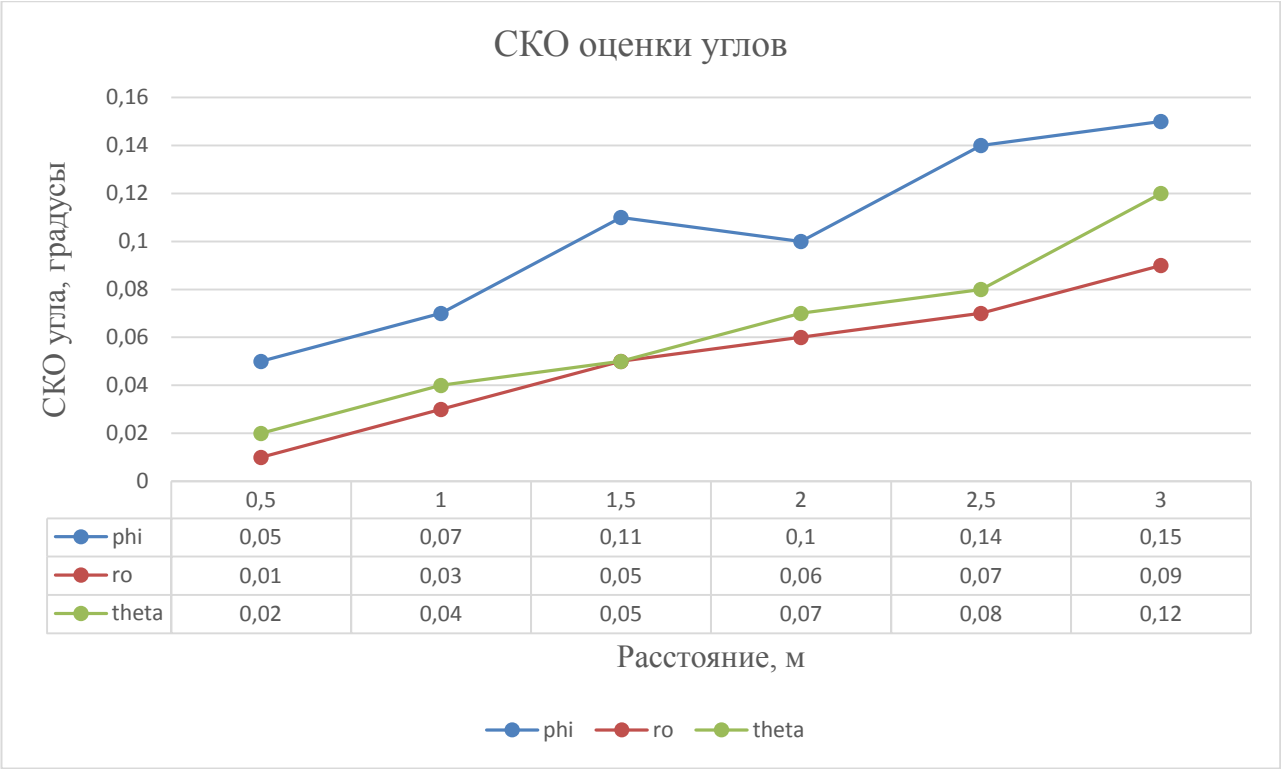


Рисунок 5.6 – Гистограмма распределения смещения по оси Z

Наибольшая дисперсия наблюдается по оси Z. Это объясняется тем, что при движении вдоль этой оси проекции точек практически не меняются.

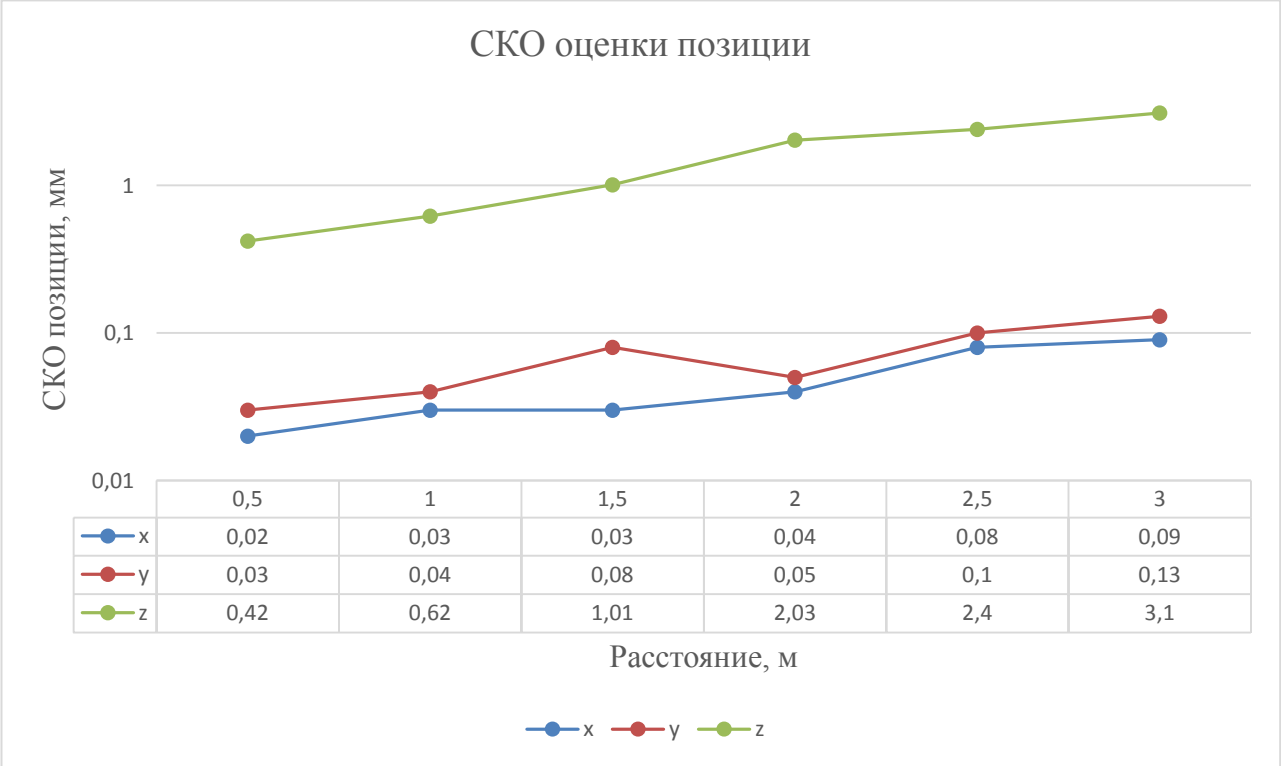
Аналогичные эксперименты были проведены на расстояниях от 0.5 до 3 метров с шагом в 0.5 метров. Результат для оценки углов приведен на графике ниже (рисунок 5.7).



	0,5	1	1,5	2	2,5	3
phi	0,05	0,07	0,11	0,1	0,14	0,15
ro	0,01	0,03	0,05	0,06	0,07	0,09
theta	0,02	0,04	0,05	0,07	0,08	0,12

Рисунок 5.7 – График СКО оценки углов в зависимости от расстояния

На рисунке 5.8 приведен график зависимости оценки позиции от расстояния.



	0,5	1	1,5	2	2,5	3
x	0,02	0,03	0,03	0,04	0,08	0,09
y	0,03	0,04	0,08	0,05	0,1	0,13
z	0,42	0,62	1,01	2,03	2,4	3,1

Рисунок 5.8 – График СКО оценки позиции в зависимости от расстояния

5.3. Оценка производительности программы

Время работы алгоритма на настольном компьютере на базе четырехъядерного процессора Inter Core i5 с частотой 3,1 ГГц. на одном кадре в среднем составляет чуть более 1 миллисекунды. На рисунке 5.9 приведена гистограмма распределения времени работы.

Рисунок 5.9 – Гистограмма распределения времени работы (i5-2400@4x3.1ГГц)

На RGB-D сенсоре, оснащенном мобильным процессором ARM-V7, скорость работы составляет порядка 4-6 мс на кадр (рисунок 5.10).

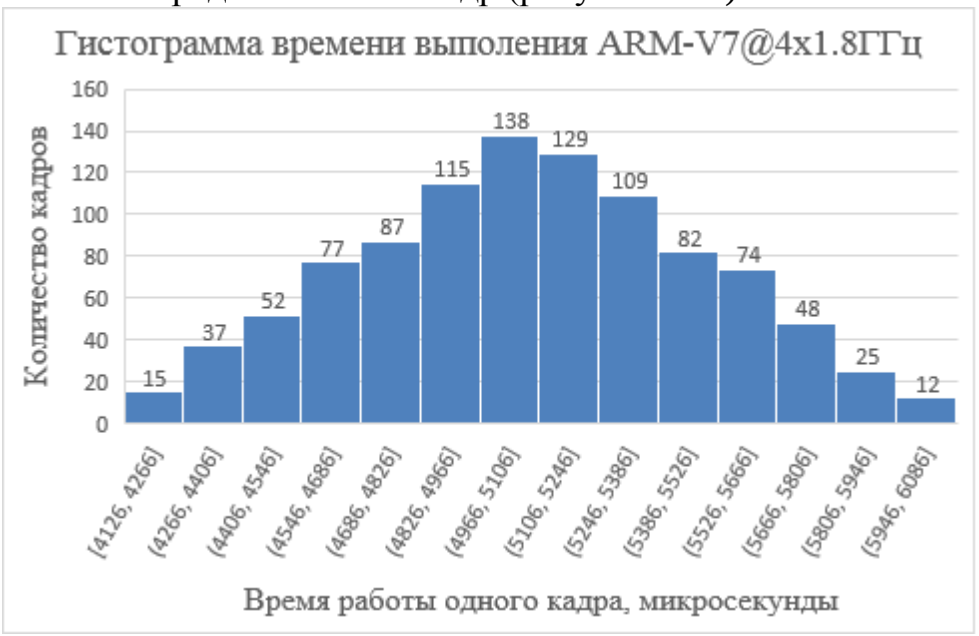


Рисунок 5.10 – Гистограмма распределения времени работы (ARM-V7@4x1.8ГГц)

Данное время работы позволяет алгоритму работать в режиме реального времени. В приложении 1 приведен фрагмент текста программы.

Выводы по разделу

В данном разделе представлены основные результаты работы. Приведены результаты замеров точности определения углов ориентации маркера в градусах. Построены гистограммы распределений. Найдены математические ожидания и среднеквадратичные отклонения оцениваемых величин.

Также приведены результаты проведенного эксперимента по определению параметров смещения. Приведены результаты определения точности смещения маркера в системе координат сцены в миллиметрах. Приведены математические ожидания и среднеквадратичные отклонения соответствующих величин. Маркер находился на расстоянии около полутора метров.

Наибольшая дисперсия наблюдается по оси Z. Это объясняется тем, что при движении вдоль этой оси проекции точек практически не меняются.

Аналогичные эксперименты были проведены на расстояниях от 0.5 до 3 метров с шагом в 0.5 метров. Приведен результат для оценки углов в виде графика. Аналогичный результат приведен для отражения зависимости оценки позиции от расстояния. В целом, по графикам можно наблюдать, что точность ухудшается с увеличением расстояния.

Произведены замеры времени работы алгоритма на настольном компьютере на базе четырехъядерного процессора Inter Core i5 с частотой 3,1 ГГц. На одном кадре в среднем оно составило чуть более 1 миллисекунды. Также приведена гистограмма распределения времени работы. На RGB-D сенсоре, оснащенном мобильным процессором ARM-V7, скорость работы составляет порядка 4-6 мс на кадр. Данное время работы позволяет алгоритму работать в режиме реального времени.

ЗАКЛЮЧЕНИЕ

В работе была исследована задача нахождения положения и ориентации жесткой модели по наблюдаемым данным с сенсора, содержащего RGB-камеру, IR-камеру, а также данные с инерционных датчиков (IMU). Данная задача является важной в области компьютерного зрения. Она возникает во многих областях, таких как робототехника, дополненная и виртуальная реальность и др.

В работе была описана математическая модель камеры и объекта. Введено понятие карты глубины. Также уделено внимание дисторсии – искажению изображения вследствие неидеальности оптических линз.

Кроме того, приведен способ устойчивой калибровки камеры. Для получения более устойчивого результата калибровки необходимо сделать снимки паттерна в разных позах, под разными углами, на разных расстояниях. Для управления этими условиями было введено четыре параметра, характеризующие каждый снимок в картинной плоскости: положение доски, доля площади доски от всего изображения, скос доски. Результатом калибровки является нахождение внутренних параметров камеры, а также дисторсии. Приведен метод обеспечения более полного покрытия пространства поз для получения более устойчивого результата калибровки.

Также была рассмотрена параметризация жесткого объекта. Под жестким понимается объект, который не деформируется, то есть относительное положение всех его точек не меняется. Объект был задан множеством его точек. Положение объекта в системе координат сенсора можно задать с помощью преобразования вращения и смещения. Приведена формула преобразования точек из системы координат объекта в систему координат картинной плоскости (проективную систему координат).

Была рассмотрена модель плоского маркера. Такие маркеры часто прикрепляются на объект, и используются для определения позы объекта. Данные маркеры просты в изготовлении (можно распечатать на обычном принтере), при этом они обладают характерной структурой, которую можно выделить на RGB-изображении. Для быстрого и эффективного определения маркера применялся поиск особых точек. Кандидаты для особых точек выбирались на основе отклика фильтра, к которому был применен метод подавления максимумов.

Для выбора нужных особых точек был построен граф, веса которого задаются исходя из критерия отличия интенсивностей по разные стороны от ребра. Из этого графа выбрался подграф, соответствующий маркеру из предположения, что сумма весов его ребер будет максимальной. В общем случае задача поиска подграфа является *NP*-полной. Поэтому для ее решения использовался направленный перебор. Из всех подграфов выбирался подграф с максимальным весом.

После нахождения особых точек их позиции были уточнены до субпиксельного уровня для получения более точного результата. Это было сделано с помощью

итеративного метода оценки моды плотности распределения отклика фильтра детектора шахматных углов.

Качество соответствия параметров модели и данных было задано с помощью ошибки репроекции. Ошибка репроекции отражает суммарное отклонение проекции точек модели при заданном вращении и смещении от проективных координат субпиксельных позиций особых точек. Данная функция оптимизировалась с помощью алгоритма Levenberg–Marquardt, который на практике показывает хорошие результаты при оптимизации квадратичных функций.

Также в работе описаны проблемы, возникающие в результате оптимизации. Первая проблема – двухмодальность оценки при нефронтальных позах. Вторая – высокая дисперсия оценки при позах, близких к фронтальным. Описано решение данных проблем с помощью использования данных с карты глубины. На облаке точек, соответствующем точкам маркера был применен метод главных компонент, позволяющий получить центр масс и нормаль, которые в дальнейшем могут быть использованы в качестве дополнительно критерия при оптимизации.

Также была рассмотрена задача трекинга шлема виртуальной. Шлемы виртуальной реальности применяются во многих областях, таких как проектирование, авиация, медицина, компьютерные игры и др. Для создания наилучшего эффекта присутствия необходимо решить задачу определения положения и ориентации шлема в пространстве. На переднюю поверхность шлема нанесен плоский маркер, который используется для трекинга.

Дополнительным фактором для улучшения качества трекинга были использованы данные о позиции ключевых точек скелета. Эти позиции известны в системе координат RGB-D сенсора. Данные с IMU датчиков же поступают в системе координат телефона. В связи с этим была сформулирована и решена задача поиска преобразования из одной системы в другую. Данный подход позволяет корректировать параметры положения и ориентации шлема, когда маркер ненаблюдаем.

Были проанализированы основные результаты работы. Приведены замеры точности определения углов ориентации маркера и позиции. Построены гистограммы распределений. Найдены математические ожидания и среднеквадратичные отклонения оцениваемых величин. Эксперименты были проведены на расстояниях от 0.5 до 3 метров с шагом в 0.5 метров. Приведены графики зависимости оценок от расстояния. По графикам можно наблюдать, что точность ухудшается с увеличением расстояния.

Произведены замеры времени работы алгоритма на настольном компьютере на базе четырехъядерного процессора Inter Core i5 с частотой 3,1 ГГц. На одном кадре в среднем оно составило чуть более 1 миллисекунды. Также приведена гистограмма распределения времени работы. На RGB-D сенсоре, оснащенном мобильным процессором ARM-V7, скорость работы составляет порядка 4-6 мс на кадр. Данные характеристики времени работы позволяют применять алгоритм в режиме реального времени.

Работы была выполнена в рамках исследования в компании 3DiVi. Алгоритм трекинга был внедрен в производство.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Rakibe, R. Background Subtraction Algorithm Based Human Motion Detection / R. Rakibe, D. Bharati // International Journal of Scientific and Research Publications. – 2013. – P. 192-197.

2 Abhishek, K. Moving Object Tracking Using Gaussian Mixture Model and Optical Flow / K. Abhishek, P. Krishan // International Journal of Advanced Research in Computer Science and Software Engineering. – 2013. – P. 152-162.

3 Athanesious, J. Systematic Survey on Object Tracking Methods in Video / J. Athanesious, P. Suresh // International Journal of Advanced Research in Computer Engineering & Technology (IJARCET). – 2012. – P. 242-247.

4 Sen-Ching, S. Robust techniques for background subtraction in urban traffic video / S. Sen-Ching, C. Kamath // Proceedings of SPIE - The International Society for Optical Engineering 5308. – 2003. – p. 158.

5 Srinivasan, L. Improved Background Subtraction Techniques for Security in Video Applications / L. Srinivasan, K. Porkumaran // 3rd ASID International Conference. – 2009. – P. 20-22.

6 Ruolin, Z. Object Tracking and Detecting Based on Adaptive Background Subtraction / Z. Ruolin, J. Ding // International Workshop on Information and Electronics Engineering. – 2012. – P. 1351-1355.

7 Athanesious J. Implementation and Comparison of Kernel and Silhouette Based Object Tracking / J. Athanesious, P. Suresh // International Journal of Advanced Research in Computer Engineering & Technology. – 2013. – P. 1298-1303.

8 Mishra, R. Multiple Object Tracking by Kernel Based Centroid Method for Improve Localization / R. Mishra, K. Mahesh // International Journal of Advanced Research in Computer Science and Software Engineering. – 2012. – P. 137-140.

9 Xiao-Shan, G. Complete solution classification for the perspective-three-point problem / G. Xiao-Shan, H. Xiao-Rong // IEEE Computer Society. – 2003. – P. 930-943.

10 Fischler, M. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography / M. Fischler, R. Bolles // Commun. of the ACM. – 1981. – P. 381-395.

11 Quan, L. Linear n-point camera pose determination / L. Quan, Z. Lan // IEEE Trans. on Pattern Analysis and Machine Intelligence. – 1999. – №21. – P. 774-780.

12 Sankari, M. Estimation of Dynamic Background and Object Detection in Noisy Visual Surveillance / M. Sankari, C. Meena // International Journal of Advanced Computer Science and Applications. – 2011. – P. 77-83.

13 Ameller, M. Camera pose revisited–new linear algorithms / M. Ameller, B. Triggs // European Conf. on Computer Vision. – 2000.

14 Horn, B. Closed-form solution of absolute orientation using unit quaternions / B. Horn // The Optical Society of America. – 1987. – №4. – P. 629-642.

15 Hartley, R. Multiple view geometry in computer vision. and A. Zisserman / R. Hartley // Cambridge Univ Press. – 2000.

16 Lepetit, V. EPnP: An Accurate O(n) Solution to the PnP Problem / V. Lepetit, M. Moreno-Noguer // International Journal of Computer Vision. 81. – 2009. – P. 155-166.

17 Sutherland, I. Three-dimensional data input by tablet / I. Sutherland // Proceedings of the IEEE. –1974. – №62. – P. 453-461.

18 Dementhon, D. Model-based object pose in 25 lines of codes / D. Dementhon, L. Davis // Int. J. of Computer Vision. – 1995. – P. 123-141.

19 Oberkampf, D. Iterative pose estimation using coplanar feature points / D. Oberkampf, L. Davis // Computer Vision and Image Understanding. – 1996. – № 63. – P. 495-511.

20 More, J. The Levenberg-Marquardt algorithm, implementation and theory / J. More // In G. A. Watson, editor, Numerical Analysis, Lecture Notes in Mathematics. – 1977. – p. 630.

21 Bennet, S. Quick and Robust Detection of Chessboard Features / S. Bennet, J. Lasenby // Computer Vision and Image Understanding. – 2001. – №81. – P. 285–292.

22 Neubeck, A. Efficient non-maximum suppression / A. Neubeck, L. Van Gool // Pattern Recognition. – 2006.

23 Saravanakumar, S. Multiple human object tracking using background subtraction and shadow removal techniques / S. Saravanakumar, A. Vadivel, A. Saneem // Signal and Image Processing (ICSIP). – 2010. – P. 79-84.

24 Lee, J. Visual tracking by partition-based histogram back projection and maximum support criteria / J. Lee, Y. Wonpil // Robotics and Biomimetics (ROBIO) IEEE International Conference. – 2011. – P. 2860-2865.

25 Welch, G. An introduction to the Kalman Filter / G. Welch, G. Bishop // In University of North Carolina at Chapel Hill, Department of Computer Science. – 2006. – P. 95-141.

26 Hitesh, A. Moving Object Tracking Using Kalman Filter / A. Hitesh, P. Darshak // International Journal of Computer Science and Mobile Computing. – 2013. – P. 326-332.

27 Shi, J. Good features to track / J. Shi, C. Tomashi // IEEE Trans. on Pattern Analysis and Machine Intelligence. – 2004. – №10. – P. 134–138.

28 Chen, D. A New Sub-Pixel Detector for X-Corners in Camera Calibration Targets / D. Chen, G, Zhang // Int. J. Computer Vision. – 1998. – №3. – P. 382–391.

29 Cheng-Laing L. Automatic path modeling by image processing techniques / L. Cheng-Laing, L. Kai-Wei // Machine Learning and Cybernetics (ICMLC). – 2010. – №5. – P. 2589-2594.

30 Karl R. Towards model-based recognition of human movements in image sequences / R. Karl // Computer Vision, Graphics, and Image Processing: Image Understanding. – 1994. – №59. – P. 94–115.

31 Hedvig S. Stochastic tracking of 3D human figures using 2D image motion / S. Hedvig, M. Black, D. Fleet // Proceedings of the European Conference on Com-puter Vision (ECCV’00), Lecture Notes in Computer Science. – 2000. – №2. – P. 702–718.

32 Delamarre Q. 3D articulated models and multiview tracking with physical forces / Q. Delamarre, F. Olivier // Computer Vision and Image Understanding. – 2001. – №81. – P. 328–357.

33 Dariu M. A 3D model-based approach tracking of humans in action / M. Da-riu, L. Gavrila, L. Davis // Proceedings of the Conference on Computer Vision and Pattern Recognition. – 1996. – P. 73–80.

34 Roland K. Markerless tracking of complex human motions from multiple views / K. Roland, L. Gool // Computer Vision and Image Understanding. – 2006. – №104. – P. 190–209.

35 Carranza J. Free-viewpoint video of human actors / J. Carranza, C. Theobalt, M. Magnor //ACM Transactions on Computer Graphics. – 2003. – №22. – P. 569–577.

36 Barro´n C. Estimating anthropometry and pose from a single uncalibrated image / C. Barro´n, I. Kakadiaris // Computer Vision and Image Understanding. – 2001. – №81. – P. 269–284.

37 Kakadiaris I. Three-dimensional human body model / I. Kakadiaris, D. Metaxas // International Journal of Computer Vision. – 1998. – №30. – P. 191–218.

38 Bottino A. A silhouette-based technique for the reconstruction of human movement / A. Bottino, A. Laurentini // Computer Vision and Image Understanding (CVIU). – 2001. – №83. – P. 79–95.

39 Pla¨nkers R. Tracking and modeling people in video sequences Pascal Fua / R. Pla¨nkers // Computer Vision and Image Understanding (CVIU). – 2001. – №81. – P. 285–302.

40 Barro´n C. Monocular human motion tracking / C. Barro´n, I. Kakadiaris // Multimedia Systems. – 2004. – №10. – P. 118–130.

41 Grauman K. Inferring 3D structure with a statistical image-based shape mod-el / K. Grauman, G. Shakhnarovich, T. Darrell // Proceedings of the International Conference on Computer Vision (ICCV'03). – 2003. – №1. – P. 641–647.

42 Chu C. Markerless kinematic model and motion capture from volume se-quences / C. Chu, O. Jenkins, M. Mataric // Proceedings of the Conference on Com-puter Vision and Pattern Recognition (CVPR'03). – 2003. – №2. – P. 475–483.

43 BenAbdelkader C. Estimation of anthropomeasures from a single calibrated camera / C. BenAbdelkader, L. Davis // Proceedings of the International Conference on Automatic Face and Gesture Recognition. – 2006. – №6. – P. 499–504.

44 Myronenko, A. Point-Set Registration: Coherent Point Drift / A. Myronenko, X. Song // IEEE Trans. on Pattern Analysis. – 2010. – №12. – P. 2262–2275.

45 Yuille, A. The motion coherence theory / A. L. Yuille, N. M. Grzywacz // Int. J. Computer Vision. – 1988. – №3. – P. 344–353.

46 Besl, P. A Method for Registration of 3-D Shapes / P. J. Besl // Pattern Anal-ysis and Machine Intelligence. – 1992. – №14. – p. 239–256.

47 Nocedal J. Numerical Optimization / J. Nocedal, S. Wright // USA, Springer. – 2006.

48 Avriel M. Nonlinear Programming: Analysis and Methods / M. Avriel // Do-ver Publishing. – 2003.

49 Reuleaux, F. The Kinematics of Machinery / F. Reuleaux // Dover, New York. – 1963. – №1. – P. 17–22.

50 Usher, P. A History of Mechanical Inventions / P. Usher // Harvard University Press. – 1968. – P. 7–10.

ПРИЛОЖЕНИЕ 1. КОД ПРОГРАММЫ

Файл MarkEstimator.h - содержит в себе объявления классов;
файл MarkEstimator.cpp - содержит методы классов.

П1.1. Файл MarkEstimator.h

```
#ifndef MARKESTIMATOR_H_
#define MARKESTIMATOR_H_

#include "markestimator_common.h"

#include <map>
#include <memory>
#include <deque>
#include <vector>
#include <iostream>

#include <opencv2/core/core.hpp>

#include "nuitrack/types/detail/SkeletonDataImpl.h"
#include "nuitrack/types/detail/RGBMarkImpl.h"
#include "MarkerFitting/AbstractMarkerFitter.h"
#include "MarkerDetection/AbstractMarkerDetector.h"
#include "DepthMarkerFitter.h"

#include "Visualization/DensityVisualizer.h"

#include "MarkerPose.h"
#include "PoseHypothesis.h"

#include <boost/property_tree/ptree.hpp>

class MarkEstimator
{
public:
    typedef std::shared_ptr<MarkEstimator> Ptr;

    MarkEstimator(const boost::property_tree::ptree &config);

    void begin(tdv::nuitrack::middleware::SkeletonDataImpl::Ptr skeletonData,
                        tdv::nuitrack::middleware::RGBMarkImpl::Ptr RGBMarks,
                        cv::Mat rgb, cv::Mat depth, cv::Mat gray,
                        float hFOV, float vFOV);

    cv::Mat _debugOutput;
```

```
private:
    void calcModel();

    bool getPose(int userId,
                                cv::Mat gray,
                                cv::Mat depth,
                                cv::Mat BGR,
                                cv::Point3f neck,
                                std::vector<MarkerPose> &poses,
                                double &errAngle,
                                cv::Point3f neckUp = cv::Point3f(0,200,0));

    double _refineTestAngle;
    std::vector<cv::Point2f> _trueFeaturesPositions;

    cv::Mat _cameraMatrix;
    std::vector<float> _distorsion;

    float _markerWidth;
    float _markerHeight;

    float _halfW;
    float _halfH;

    float _hFOV;
    float _vFOV;

    std::vector<cv::Point3f> _modelPoints3D;

    AbstractMarkerDetector::Ptr _markerDetector;
    AbstractMarkerFitter::Ptr _markerFitter;
    DepthMarkerFitter::Ptr _depthMarkerFitter;

    std::shared_ptr<tdv::nuitrack::NuitrackCoordConverter> _depthCoordConverter;

    std::map<int, std::vector<PoseHypothesis> > _poseHypotheses;

    struct HypothesesInfo
    {
            HypothesesInfo():numFails(0), numSamples(0){}

            HypothesesInfo(int numFailes, int numSamples):
                    numFails(numFails), numSamples(numSamples){}

            int numFails;
            int numSamples;
    };

    std::map<int, HypothesesInfo> _hypothesesInfo;
    float _ZPD_depth;
```

```
    float _ZPD_rgb;
    float _scaleRGBCoef;
    cv::Rect _roiRect;
    bool _depthPoseAmbiguitySolving;
    bool _mirror;
    int _numFrame;
    bool _cameraMatrixScaled;
#ifdef MARK_ESTIMATOR_INTERNAL_DEBUG
    std::vector< std::deque<double> > _positionsX;
    std::vector< std::deque<double> > _positionsY;
    std::vector< std::deque<cv::Point2f> > _positions;
    std::deque<double> _phiAngles;
    std::deque<double> _thetaAngles;
    std::deque<double> _roAngles;
    std::deque<double> _xTransl;
    std::deque<double> _yTransl;
    std::deque<double> _zTransl;
    std::deque<double> _normalX;
    std::deque<double> _normalY;
    std::deque<double> _normalZ;

    DensityVisualizer _dv;
#endif
    std::vector<cv::Point2f> _corrPoints;
    enum DIFF_TYPE
    {
        COMPARE_WITH_IDENTITY,
        COMPARE_ANGLE_BETWEEN_NORMALS
    };
    float getMatDiff(cv::Mat A, cv::Mat B, DIFF_TYPE dt);
    cv::Point3f applyPose(const cv::Point3f& point, const MarkerPose& pose);
    void makeMarkerMat();
    bool updateHyposAndGetAnswer(const int skeletonId,
                                 std::vector<MarkerPose>& poses,
                                 cv::Mat& rot,
                                 cv::Mat& trans);
    void updateHypotheses(int skeletonId, std::vector<MarkerPose> &poses);
    MarkerPose getBestHypothesisPose(int skeletonId);
    void smoothSlerp(const cv::Mat& pRot,
                     const cv::Mat& hRot,
                     const float alphaRot,
                     cv::Mat& rotSmoothed);
    void showCubic(const cv::Mat &rgb,
                   const cv::Mat &depth,
                   const cv::Mat &rot,
                   const cv::Mat &trans,
                   bool gotAnswer,
                   double errAngle,
                   int skeletonId);
    void showAngleHist(const cv::Mat &rot, const cv::Mat &trans, bool gotAnswer);
    void showHypothesesAngles(const cv::Mat &rot, int skeletonId);
```

```
    void refineSyntheticTest(cv::Mat& rgb, cv::Mat& gray, cv::Point3f &neck3f);
    void reprojectionMatrixSynthTest();
    void synthRGBTest();
    void featuresHists(const std::vector<cv::Point2f> &corrPoints);
};
#endif
```

П1.2. Файл MarkEstimator.cpp

```
#include "Utils/SyntheticMarkerAccuracyChecker.h"
#include "Utils/MarkerRenderer.h"
#ifdef MARK_ESTIMATOR_INTERNAL_DEBUG
    #ifdef NORMALIZE_INTENSITY
            #define EQUALIZE_HIST
//          #define SHARPEN
            #ifdef MARK_ESTIMATOR_INTERNAL_DEBUG
    //              #define SHOW_NORMALIZED
            #endif
    #endif
//#define RGBDFITTER
using namespace std;
using namespace cv;
using MarkerPoseEstimation::scalePoint2f;
using boost::property_tree::ptree;
using tdv::nuitrack::NuitrackCoordConverter;
MarkEstimator::MarkEstimator(const ptree &config)
{
    _markerWidth = config.get<double>("RGBMarkEstimator.MarkerWidth");
    _markerHeight = config.get<double>("RGBMarkEstimator.MarkerHeight");
    _depthPoseAmbiguitySolving = true;
    _mirror = false;
    this->calcModel();
    _numFrame = 0;
    _refineTestAngle = 0;
    _trueFeaturesPositions.clear();
    const float fx_rgb = 5.1562081360466846e+02;
    const float fy_rgb = 5.1562081360466846e+02;
    float cx_rgb = 3.2310080819898241e+02;
    float cy_rgb = 2.4583484136480163e+02;
    _cameraMatrix = Mat(3,3, CV_64F);
    _cameraMatrix.setTo(0.0);
    _cameraMatrix.at<double>(0,0) = fx_rgb;
    _cameraMatrix.at<double>(1,1) = fy_rgb;
    _cameraMatrix.at<double>(2,2) = 1.0f;
    _cameraMatrix.at<double>(0,2) = cx_rgb;
    _cameraMatrix.at<double>(1,2) = cy_rgb;
    _cameraMatrixScaled = false;
    _distorsion = vector<float>();
//  _distorsion = { 2.8961708018597021e-01,
```

```
    //                                  -1.1838676704195323e+00,
    //                                  0.0,
    //                                  0.0,
    //                                  1.5844044223473550e+00};
    #ifdef MARK_ESTIMATOR_INTERNAL_DEBUG
        _positionsX.resize(9); // number of features on marker
        _positionsY.resize(9); // number of features on marker
        _positions.resize(9); // number of features on marker
    #endif
    }
    void MarkEstimator::begin(tdv::nuitrack::middleware::SkeletonDataImpl::Ptr skeletonData,
                                        tdv::nuitrack::middleware::RGBMarkImpl::Ptr RGB-
Marks,
                                        cv::Mat rgb, cv::Mat depth, cv::Mat gray,
                                        float hFOV, float vFOV)
    {
    //  waitKey();
        SCOPED_PERF()
        _hFOV = hFOV;
        _vFOV = vFOV;
        _ZPD_depth = 0.5f / tan(_hFOV * 0.5f) * depth.cols;
        _ZPD_rgb = 0.5f / tan(_hFOV * 0.5f) * rgb.cols;
        _scaleRGBCoef = rgb.cols / 640.0f;
        if (!_cameraMatrixScaled)
        {
                _cameraMatrix.at<double>(0,0) *= _scaleRGBCoef;
                _cameraMatrix.at<double>(1,1) *= _scaleRGBCoef;
                _cameraMatrix.at<double>(0,2) = _scaleRGBCoef * (_cameraMatrix.at<double>(0,2) +
0.5) - 0.5;
                _cameraMatrix.at<double>(1,2) = _scaleRGBCoef * (_cameraMatrix.at<double>(1,2) +
0.5) - 0.5;
                _cameraMatrixScaled = true;
        }
        if (!_depthCoordConverter)
        {
                _depthCoordConverter =
                        std::make_shared<NuitrackCoordConverter>(hFOV,

         vFOV,

         depth.size(),

         _mirror);
        }
        if (!_markerFitter)
        {
    #ifndef RGBDFITTER
                _markerFitter = std::make_shared<PNPFitter>();
    //          _markerFitter = std::make_shared<RPPFitter>();
    //          _markerFitter = std::make_shared<RGBDFitter>(_hFOV, _vFOV, _mirror);
    #endif
```

```
    #ifdef RGBDFITTER
            _markerFitter = std::make_shared<RGBDFitter>();
    #endif
        }
        if (!_depthMarkerFitter)
        {
            _depthMarkerFitter = std::make_shared<DepthMarkerFitter>();
        }
        if (!_markerDetector)
        {
    #if MARKER_TYPE  == CHESS_CROSS_MARKER
            _markerDetector = std::make_shared<ChessCrossMarkerDetector>(_mirror);
    #endif
    #if MARKER_TYPE  == CHESSBOARD_MARKER
            _markerDetector = std::make_shared<ChessboardDetector>(14,9);
    #endif
        }
    #ifdef MARK_ESTIMATOR_INTERNAL_DEBUG
        _debugOutput = rgb.clone();
    #endif
    #ifndef FRONTAL_VIEW_DEBUG
        for (auto skeleton : skeletonData->getSkeletons())
    #endif
        {
    #ifndef FRONTAL_VIEW_DEBUG
            int skeletonId = skeleton.id;
            tdv::nuitrack::Vector3 neck = skeleton.joints[ tdv::nuitrack::JointType::JOINT_NECK
].real;
            cv::Point3f neck3f(neck.x,neck.y,neck.z);
    #else
            int skeletonId = 1;
            cv::Point3f neck3f(0, -200,    2000);
    #endif
            vector<MarkerPose> poses;
            double errAngle;
    //#define REFINE_SYNTH
    #ifdef REFINE_SYNTH
            this->refineSyntheticTest(rgb, gray, neck3f);
    #endif
            bool foundPose = this->getPose(skeletonId, gray, depth, rgb, neck3f, poses, errAngle);
            if (!foundPose)
                poses.clear();
            Mat rot;
            Mat trans;
            START_PERF("updateHyposAndGetAnswer")
            bool gotAnswer = this->updateHyposAndGetAnswer(skeletonId, poses, rot, trans);
            STOP_PERF("updateHyposAndGetAnswer")
            if (!gotAnswer)
            {
                rot = Mat(3,3, CV_32F);
                rot.setTo(0);
```

```
                trans = Mat(3,1, CV_32F);
                trans.setTo(0);
        }
#ifdef SHOW_TRANSLATION
        if (gotAnswer)
        {
                std::cout << trans << endl;
        }
#endif
#ifdef SHOW_ANGLES_HIST
        this->showAngleHist(rot, trans, gotAnswer);
#endif
#ifdef SHOW_CUBIC
        this->showCubic(rgb, depth, rot, trans, gotAnswer, errAngle, skeletonId);
#endif
#ifdef SHOW_HYPS
        this->showHypothesesAngles(rot, skeletonId);
#endif
        rot.convertTo(rot, CV_32F);
        trans.convertTo(trans, CV_32F);
        RGBMarks->setRotationMat(rot.ptr<float>(0),skeletonId);
        RGBMarks->setTranslationMat(trans.ptr<float>(0),skeletonId);
        RGBMarks->setErrAngle(errAngle, skeletonId);
    }
}
RNG rng(getTickCount());
bool MarkEstimator::getPose(int userId,
                                                Mat gray,
                                                Mat depth,
                                                Mat rgb,
                                                Point3f neck,
                                                vector<MarkerPose>& poses,
                                                double &errAngle,
                                                Point3f neckUp)
{
    SCOPED_PERF()
//#define REPROJ_MAP
#ifdef REPROJ_MAP
    this->reprojectionMatrixSynthTest();
#endif
//#define SYNTH_RGB_TEST
#ifdef SYNTH_RGB_TEST
    this->synthRGBTest();
#endif
#ifdef SHOW_DEPTH_MAP
    imshow("depth", depth * 8);
#endif
#ifdef SHOW_NUM_FRAME
    std::cout << "Frame: " << _numFrame << std::endl;
#endif
    ++_numFrame;
```

```
        poses.clear();
    #ifdef SHOW_HEAD_DEPTH
        std::cout << "User #" << userId <<" CURRENT HEAD DEPTH: " << z << std::endl;
    #endif
        if(neck.z < _ZPD_depth)
                return 0;
        float roiHeight = 400.0f; //mm
        float roiWidth = 400.0f; //mm
    #if MARKER_TYPE == CHESSBOARD_MARKER
        roiHeight  = 400000;
        roiWidth = 400000;
    #endif
        auto convertMmToPixels = [=](float mm) -> float
                {return mm * _ZPD_rgb / neck.z;};
        int vSize = convertMmToPixels(roiHeight); //ROI height
        int hSize = convertMmToPixels(roiWidth);  //ROI width
        Rect screen(Point2i(0, 0), gray.size());
        //coordinated in depth camera system
        Point3f shiftedNeck_Depth_CamCS = neck + neckUp;
        cv::Mat shiftedNeckVec = (cv::Mat_<double>(3, 1) <<
                                                        shiftedNeck_Depth_CamCS.x,
                                                        -shiftedNeck_Depth_CamCS.y,
                                                        shiftedNeck_Depth_CamCS.z);
        cv::Mat shiftedNeckProjHomogen = _cameraMatrix * shiftedNeckVec;
        Point2f initMarkerPointProj(shiftedNeckProjHomogen.at<double>(0,0) /
                                                        shiftedNeckProjHomogen.at<double>(2,0),
                                                        shiftedNeckProjHomogen.at<double>(1,0)
/

        shiftedNeckProjHomogen.at<double>(2,0));
        Rect roiRect(initMarkerPointProj.x - hSize/2,
                                initMarkerPointProj.y - vSize/2,
                                hSize,
                                vSize);
    #ifdef MARK_ESTIMATOR_INTERNAL_DEBUG
        cv::rectangle(_debugOutput,roiRect,Scalar(0,255,0));
    //  cv::circle(debugOutput, 0.5 * Point2f(gray.cols, gray.rows), 2, CV_RGB(0,255,0),-1);
    #endif
        roiRect &= screen;
        _roiRect = roiRect;
        if (roiRect.width < 16 || roiRect.height < 16)
        {
                #ifdef MARK_ESTIMATOR_INTERNAL_DEBUG
                std::cout << "Roi rectangle is less then 16 pixels!:\n";
                #endif
                return false;
        }
        Mat miniRGB = rgb(roiRect);
        Mat miniGray = gray(roiRect);
        Mat miniMorphGray;
    #ifdef DO_MORPHOLOGY
```

```
        START_PERF("morphology")
        morphologyEx(miniGray, miniMorphGray, CV_MOP_GRADIENT, Mat());
        STOP_PERF("morphology")
        #ifdef SHOW_MORPHOLOGY
        Mat morphZoomIn;
        cv::resize(miniMorphGray, morphZoomIn, Size(), 640.0f / roiRect.width, 640.0f / roiRect.width, INTER_NEAREST);
        cv::imshow("morphGray " + to_string(userId), morphZoomIn);
        #endif
    #endif
        vector<Point2f> corrPoints;
        START_PERF("DetectMarker")
        bool detected = _markerDetector->detect(userId, roiRect, miniGray, miniMorphGray, miniRGB, _ZPD_rgb, neck, corrPoints);
        STOP_PERF("DetectMarker")
        if (!detected)
            return false;
    #ifdef FEATURES_HISTS
        this->featuresHists(corrPoints);
    #endif
        for (auto& corrPoint : corrPoints)
            corrPoint += Point2f(roiRect.x, roiRect.y);
        _corrPoints = corrPoints;
        MarkerPose curPose;
        MarkerPose secondPose;
        bool startFromPrevPose = false;
        START_PERF("PNP")
        bool result = _markerFitter->fit(
                                                corrPoints,
                                                _modelPoints3D,
                                                _cameraMatrix,
                                                _distorsion,
                                                curPose,
                                                depth,
                                                rgb,
                                                startFromPrevPose);
        STOP_PERF("PNP")
        if (!result)
        {
    #ifdef MARK_ESTIMATOR_INTERNAL_DEBUG
            std::cout << "PNP FAILED!\n";
    #endif
            return false;
        }
        MarkerPose pnpPose = curPose;
    #ifdef RGBDFITTER
        secondPose = curPose;
    #else
        bool isFrontal = false;
    #ifdef FRONTAL_REJECTION
        Mat curTransl = curPose.getTranslation(); //3x1
```

```
curTransl /= norm(curTransl);
```

```
        Mat curNormal = curPose.getRotation().col(2); //3x1;
        double curDot = curTransl.dot(curNormal);
        const double frontalAngleThresh = 15.0/*degree*/;
        double curDotUpThresh = cos (frontalAngleThresh  * CV_PI / 180.0);
        if (curDot > curDotUpThresh)
        {
                isFrontal = true;
        }
    #endif
                        Mat t1 = curPose.getTranslation();
                        cv::solvePnP(_modelPoints3D, corrPoints, _cameraMatrix, _distorsion, r1, t1,
true);
                        Mat r2 = secondPose.getRotationVec();
                        Mat t2 = secondPose.getTranslation();
                        cv::solvePnP(_modelPoints3D, corrPoints, _cameraMatrix, _distorsion, r2, t2,
true);
        }
    #endif
        curPose.calcReprojectionError(_modelPoints3D, corrPoints, _cameraMatrix);
        secondPose.calcReprojectionError(_modelPoints3D, corrPoints, _cameraMatrix);
        poses.push_back(curPose);
        poses.push_back(secondPose);
        STOP_PERF("Second pose calc")
    #ifdef SHOW_RESULT_PROJECTED_POINTS
        std::vector<cv::Point2f> projectedPoints;
        cv::projectPoints(_modelPoints3D,      curPose.getRotationVec(),      curPose.getTranslation(),
_cameraMatrix, vector<float>(), projectedPoints);
        for (auto& p : projectedPoints)
        {
                circle(_debugOutput, p, 2, Scalar(0,0,255), -1);
        }
    #endif
        return result;
    }
```