|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА - Российский технологический университет»**  **РТУ МИРЭА** |

Институт искусственного интеллекта

Кафедра проблем управления

**ОТЧЁТ по ТЕХНОЛОГИЧЕСКОЙ ПРАКТИКЕ**

**Тема практики:** **Моделирование и оценка точности визуальной навигации мобильного робота в виртуальной среде**

приказ университета о направлении на практику

от «\_\_» \_\_\_\_\_\_ \_\_\_\_\_ г. № \_\_\_\_

|  |  |  |
| --- | --- | --- |
| Отчет представлен к рассмотрению: |  |  |
| Студент группы  КРБО-03-22 | \_\_\_\_\_\_\_\_\_\_\_\_ | Е.О. Попов  «\_\_» \_\_\_\_\_\_ 202\_ г. |
|  |  |  |
| Отчет утвержден |  |  |
| Допущен к защите: |  |  |
|  |  |  |
| Руководитель практики от кафедры | \_\_\_\_\_\_\_\_\_\_\_\_ | С.А.К. Диане  «\_\_» \_\_\_\_\_\_ 202\_ г. |
|  |  |  |
| Руководитель практики от университета | \_\_\_\_\_\_\_\_\_\_\_\_ | А.А. Сухоленцева  «\_\_» \_\_\_\_\_\_ 202\_ г. |

Москва 2025

**ОТЧЕТ**

**по технологической практике**

**студента 3 курса учебной группы КРБО-03-22**

**института искусственного интеллекта**

**Попова Егора Олеговича**

Практику проходил с 10.02.2025 по 06.06.2025 в лаборатории для самостоятельных занятий кафедры проблем управления

1. Задание на практику выполнил в полном объеме.
2. Подробное содержание выполненной на практике работы и достигнутые результаты: изучены исследования в области визуальной о; проведено ознакомление со средой моделирования CoppeliaSim, методами локализации.
3. Предложения по совершенствованию организации и прохождения практики: предложений нет.

Студент \_\_\_\_\_\_\_\_\_\_\_\_ (Попов Е.О.) «\_\_» \_\_\_\_\_\_\_\_ 2025 г.

1. Заключение руководителя от кафедры проблем управления по практике студента Попова Егора Олеговича:

В ходе практики, Попов Е.О. приобрел следующие профессиональные компетенции:

- способность анализировать существующие алгоритмы визуальной навигации;

- умение разрабатывать программное обеспечение в сфере визуальной навигации в виртуальной ссреде;

- готовность участвовать в составлении аналитических обзоров и научно-технических отчетов по результатам выполненной работы, в подготовке публикаций по результатам исследований и разработок.

Попов Е.О. проявил себя как организованный, целеустремленный специалист, способный самостоятельно решать поставленные перед ним инженерно-технические задачи и представить отчет по практике в полном объеме и в установленный срок и заслуживает оценки «\_\_\_\_\_\_\_\_\_\_».

Руководитель практики от кафедры     \_\_\_\_\_\_\_\_\_\_\_\_    С.А.К. Диане

«\_\_» \_\_\_\_\_\_ 202\_ г.

Руководитель практики от университета \_\_\_\_\_\_\_\_\_\_   А.А. Сухоленцева

«\_\_» \_\_\_\_\_\_ 202\_ г.

|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА - Российский технологический университет»**  **РТУ МИРЭА** |

Институт искусственного интеллекта

*(наименование института, филиала)*

Кафедра проблем управления

*(наименование кафедры)*

**ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА ТЕХНОЛОГИЧЕСКУЮ ПРАКТИКУ**

*(указать вид практики*)

**Студенту 3 курса учебной группы КРБО-03-22**

**Попову Егору Олеговичу**

**Место и время практики: лаборатория для самостоятельных занятий кафедры проблем управления Г-208, 10.02.2025 - 06.06.2025**

**Должность на практике:** -

**1. ЦЕЛЕВАЯ УСТАНОВКА: Моделирование и оценка точности визуальной навигации мобильного робота в виртуальной среде**

**2. СОДЕРЖАНИЕ ПРАКТИКИ:**

2.1 Изучить: Метод ORB для выделения ключевых точек на видеоизображении с камеры робота; расчетные формулы прямого и обратного проективного преобразования для монокулярных камер

2.2 Практически выполнить: разработать виртуальную модель местности и автономного шагающего робота; реализовать тестовое движение робота по местности; реализовать визуальную навигацию робота по камере

2.3 Ознакомиться: Со средой моделирования “CoppeliaSim, программными библиотеками OpenCV, ORBSLAM\_3, методами локализации ориентиров в поле зрения камеры с помощью фильтров Калмана и частиц

**3.ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ:** -

**4. ОРГАНИЗАЦИОННО-МЕТОДИЧЕСКИЕ УКАЗАНИЯ:** -

Заведующий кафедрой:

«\_\_»\_\_\_\_\_\_\_\_\_\_202\_\_г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.П. Романов

СОГЛАСОВАНО:

Руководитель практики от кафедры:

«\_\_»\_\_\_\_\_\_\_\_\_\_202\_\_г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.А.К. Диане

Руководитель практики от Университета:

«\_\_»\_\_\_\_\_\_\_\_\_\_202\_\_г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Сухоленцева

Задание получил:

«\_\_»\_\_\_\_\_\_\_\_\_\_202\_\_г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.О. Попов

**Проведенные инструктажи:**

Охрана труда: «\_\_»\_\_\_\_\_\_\_\_\_202\_\_г.

Инструктирующий \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ доцент С.А.К. Диане

*Подпись Расшифровка, должность*

Инструктируемый \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.О. Попов

*Подпись Расшифровка*

Техника безопасности: «\_\_»\_\_\_\_\_\_\_\_\_202\_\_г.

Инструктирующий \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ доцент С.А.К. Диане

*Подпись Расшифровка, должность*

Инструктируемый \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.О. Попов

*Подпись Расшифровка*

Пожарная безопасность: «\_\_»\_\_\_\_\_\_\_\_\_202\_\_г.

Инструктирующий \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ доцент С.А.К. Диане

*Подпись Расшифровка, должность*

Инструктируемый \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.О. Попов

*Подпись Расшифровка*

С правилами внутреннего распорядка ознакомлен: «\_\_»\_\_\_\_\_\_\_\_\_202\_\_г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.О. Попов

*Подпись Расшифровка*

|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА - Российский технологический университет»**  **РТУ МИРЭА** |

**РАБОЧИЙ ГРАФИК ПРОВЕДЕНИЯ**

**ТЕХНОЛОГИЧЕСКОЙ ПРАКТИКИ**

студента 3 курса группы КРБО-03-22 очной формы обучения, обучающегося по направлению подготовки «Мехатроника и робототехника», профиль «Автономные роботы »

|  |  |  |  |
| --- | --- | --- | --- |
| **Неделя** | **Сроки выполнения** | **Этап** | **Отметка о выполнении** |
| **1-5** | **10.02.2025-10.03.2025** | **Изучение дескриптора ORB и методов эпиполярной геометрии** |  |
| **5-10** | **10.03.2025-10.04.2025** | **Ознакомление со средой моделирования CoppeliaSim, методами локализации ориентиров** |  |
| **10-16** | **10.04.2025-06.06.2025** | **Программная реализация системы визуальной навигации и анализ накопления ошибки** |  |

**Согласовано:**

|  |  |  |
| --- | --- | --- |
| Заведующий кафедрой | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | М.П. Романов |
| Руководитель практики от кафедры | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | С.А.К. Диане |
| Руководитель практики от Университета | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | А.А. Сухоленцева |
| Обучающийся | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Е.О. Попов |

**Оглавление**

[Аннотация 7](#_Toc199882760)

[Глава 1. Изучение метода ORB для извлечения ключевых точек, расчетных формул прямого и обратного проективного преобразования для монокулярных камер. 8](#_Toc199882761)

[1.1 Метод ORB 8](#_Toc199882762)

[1.2 Прямое и обратное проективные преобразования для монокулярных камер 10](#_Toc199882763)

[2.1 Среда моделирования CoppeliaSim 14](#_Toc199882765)

[2.2 Локализация ориентиров в поле зрения с помощью фильтра Калмана. 15](#_Toc199882766)

[2.3 Локализация ориентиров в поле зрения с помощью фильтра частиц. 17](#_Toc199882767)

[Глава 3. Разработка виртуальной модели местности и автономного шагающего робота, реализация тестового движения робота по местности и навигации робота по камере. 19](#_Toc199882768)

[3.1 Модель платформы и окружающей среды 19](#_Toc199882769)

[3.2 Восстановление траектории движения 22](#_Toc199882770)

[3.3 Применение метода ORB для поиска точек и вычисления дескрипторов 23](#_Toc199882771)

[3.4 Сопоставление дескрипторов и выделение наиболее точных признаков с помощью теста Лоу 23](#_Toc199882772)

[3.5 Преобразование точек в нормализованное пространство устройства посредством умножения на обратную калибровочную матрицу камеры 24](#_Toc199882773)

[3.6 Вычисление основной матрицы по особым точкам двух кадров 25](#_Toc199882774)

[3.7 Декомпозиция основной матрицы 27](#_Toc199882775)

[3.9 Результат восстановления траектории по видео полученному во время тестового движения 28](#_Toc199882776)

[Список использованной литературы 35](#_Toc199882777)

# **Аннотация**

Для достижения достаточно высокой точности позиционирования в задачах навигации повсеместно применяются алгоритмы работающие по принципу одновременной локализации и картографирования (SLAM), процедура сопоставления получаемых данных сенсоров с картой является вычислительно сложной, поэтому многие современные алгоритмы (Hector SLAM, Graph Slam и т.д.) базируются на допущении что агент перемещается в одной плоскости и для его локализации хватит 2D карты.

По техническим причинам данное допущение не применимо к задачам визуальной навигации, они требуют наличия трехмерной карты сопоставление с которой заметно сложнее чем с двумерной [1]. Поэтому сверку с картой в процессе навигации производят не при каждом изменении положения, а в моменты, когда ожидается существенное накопление ошибки [2]. В рамках данной работы реализована система визуальной навигации и проведено исследование накопления ошибки позиционирования шагающей платформы при использовании метода монокулярной навигации на базе ORB-дескриптора для возможности оптимизации работы SLAM-ориентированных методов навигации подобных платформ с заданной точностью в дальнейшем.

# **Глава 1. Изучение метода ORB для извлечения ключевых точек, расчетных формул прямого и обратного проективного преобразования для монокулярных камер.**

В сфере визуальной навигации поиск и сопоставление ключевых точек на изображениях является крайне распространенной процедурой, на эту тему опубликовано множество научных статей [8]. Одним из наиболее популярных методов является ORB (Oriented FAST Rotated BRIEF).

# **1.1 Метод ORB**

Как следует из названия в основе данного метода поиска и сопоставления особых точек лежит извлечение особых точек с помощью ориентированного FAST детектора и вычисление дескрипторов данных точек методом BRIEF.

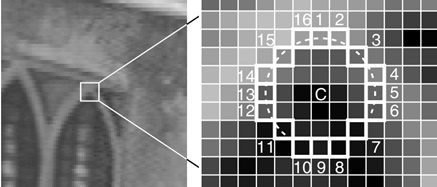
FAST, впервые предложенный в 2006 году, был одним из первых эвристических методов поиска особых точек, который завоевал большую популярность из-за своей вычислительной эффективности [9]. Для принятия решения о том, считать заданную точку С особой или нет, в этом методе рассматривается яркость пикселов на окружности с центром в точке С и радиусом 3.

Рис.1 Принцип оценки окрестности точки С FAST- детектором

Сравнивая яркости пикселов окружности с яркостью центра C, получаем для каждого три возможных исхода (светлее, темнее, похоже):

Здесь I – яркость пикселов,

t – некоторый заранее фиксированный порог по яркости.

Точка помечается как особая, если на круге существует подряд n=12 пикселов, которые темнее, или 12 пикселов, которые светлее, чем центр.

Как показала практика, в среднем для принятия решения нужно было проверить около 9 точек [10]. Для того, чтобы ускорить процесс, авторы предложили сначала проверить только четыре пиксела под номерами: 1, 5, 9, 13. Если среди них есть 3 пиксела светлее или темнее, то выполняется полная проверка по 16 точкам, иначе – точка сразу помечается как «не особая». Это сильно сокращает время работы, для принятия решения в среднем достаточно опросить всего лишь около 4 точек окружности.

FAST сам по себе не предоставляет информацию об ориентации. ORB добавляет компонент ориентации к FAST, делая его инвариантным к вращению.

Дескриптор BRIEF представляется в виде вектора длиной 256,

состоящего из результатов бинарных тестов вокруг особой точки. В

окрестности 31 × 31 пиксель сравниваются средние значения яркостей между

𝑥 и 𝑦, где 𝑥, 𝑦 – области 5 × 5 пикселей [11]:

Для достижения инвариантности к вращению область вычисления дескриптора ориентируется по ориентации особой точки 𝜃. Все 𝑛 = 256 наборов 𝑥𝑖 и 𝑦𝑖 формируют матрицу 𝑆 размерностью 2 × 𝑛. Далее 𝑆 с помощью матрицы поворота 𝑅𝜃 ориентируется в соответствии с углом 𝜃:

А сам вектор дескриптора записывается в виде:

ORB использует модифицированную версию дескриптора BRIEF (Binary Robust Independent Elementary Features), называемую rBRIEF. Данный алгоритм оптимизирован для инвариантности вращения и вычисляется в системе координат, выровненной по ориентации [3].

Простота и небольшая вычислительная сложность BRIEF, а также инвариантность детектора FAST к повороту и смене освещенности делают ORB наиболее применимым алгоритмом в области навигации из-за и без того больших объемов вычислений в реальном времени [12].

# **1.2 Прямое и обратное проективные преобразования для монокулярных камер**

Проективное отображение двумерной плоскости в себя является основной моделью при описании геометрических свойств изображений пространственных сцен на телевизионных или фотографических кадрах. При этом, проективные параметры обычно описываются в виде элементов матриц

однородных координат и их линейных преобразований. Здесь хорошо

известны постановки и решения задач трёх типов: задач машинной графики,

задач калибровки видеокамер в интересах фотограмметрии и задач управления

движением робота.

В задачах машинной графики для получения в рамках геометрической оптики параметров матрицы проективного отображения рисуемой трехмерной сцены, как правило, задаются значения фокусного расстояния объектива и координаты его расположения в системе координат сцены. Однако следует отметить, что точные параметры расположения в пространстве источника видеоизображений и его угла зрения, как правило, неизвестны, или известны лишь приближённо. Вместе с этим, в искомом проективном преобразовании сцен гораздо более важную роль часто играют факторы нарушений постулатов геометрической оптики. Это обстоятельство породило обширную литературу, связанную с решением задач калибровки видеокамер по предъявляемым образцам сцен с известными реперными точками. Наиболее ранние из этих работ были связаны с проблемами астрометрии и аэрофотосъёмки, когда высокоточная коррекция дисторсии и астигматизмов оптики играла наиболее важную роль в метрических интерпретациях деталей получаемых сцен [13].

Современные CCD-камеры хорошо описываются с помощью следующей модели, называемой проективной камерой (projective camera, pinhole camera). Проективная камера определяется центром камеры, главной осью — лучом начинающимся в центре камеры и направленным туда, куда камера смотрит, плоскостью изображения — плоскостью на которую выполняется проецирование точек, и системой координат на этой плоскости. В такой модели, произвольная точка пространства X проецируется на плоскость изображения в точку x лежащую на отрезке CX, который соединяет центр камеры C с исходной точкой X (см. рис. 2).

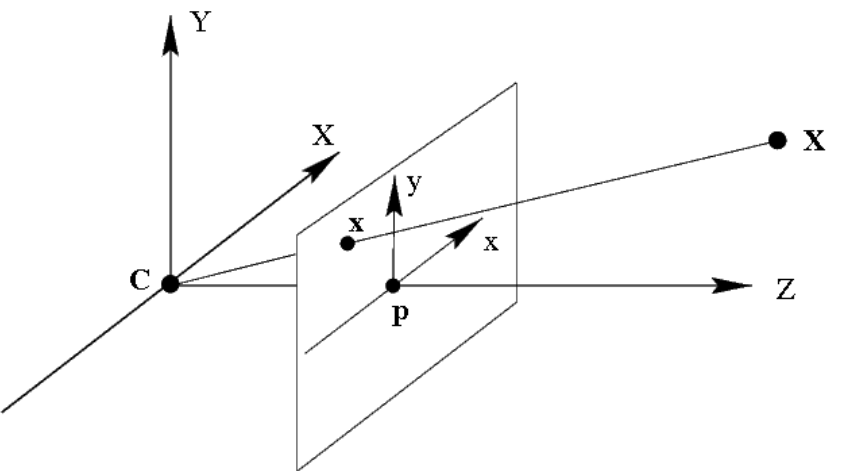


Рис.2 Модель камеры. C — центр камеры, Cp — главная ось камеры. Точка X трехмерного пространства проецируется в точку x — на плоскости изображения.

Формула проецирования имеет простую математическую запись в однородных координатах:

соответственно формула обратного проецирования в однородных координатах:

где — однородные координаты точки пространства, — однородные координаты точки плоскости, — матрица камеры размера 3 на 4.

Матрица выражается следующим образом:

где K — верхняя треугольная матрица внутренних параметров камеры размера 3 на 3 (конкретный вид приведен ниже), R — ортогональная матрица размера 3 на 3, определяющая поворот камеры относительно глобальной системы координат, I — единичная матрица размера 3 на 3, вектор c — координаты центра камеры,.

В самом простейшем случае, когда центр камеры лежит в начале координат, главная ось камеры сонаправленна оси Cz, оси координат на плоскости камеры имеют одинаковый масштаб (что эквивалентно квадратным пикселям), а центр изображения имеет нулевые координаты, матрица камеры будет равна P = K[I|0], где:

У реальных CCD камер пикселы обычно незначительно отличаются от квадратных, а центр изображения имеет ненулевые координаты. В таком случае матрица внутренних параметров примет вид:

Коэффициенты f, x, y — называются фокусными расстояниями камеры (соответственно общим и вдоль осей x и y)[14-15].

# **Глава 2. Ознакомление со средой моделирования и методами локализации ориентиров.**

# **2.1 Среда моделирования CoppeliaSim**

**CoppeliaSim**, ранее известный как V-REP, представляет собой робототехнический симулятор, используемый в промышленности, образовании и исследованиях. Первоначально она была разработана в рамках отдела исследований и разработок Toshiba и в настоящее время активно разрабатывается и поддерживается Coppelia Robotics AG, небольшой компанией, расположенной в Цюрихе, Швейцария.

Он построен на основе распределенной архитектуры управления, включающей скрипты Python и Lua или плагины C/C++, действующие как отдельные синхронные контроллеры. Дополнительные асинхронные контроллеры могут выполняться в другом процессе, потоке или машине через различные решения промежуточного программного обеспечения (ROS, remote API, ZeroMQ) с такими языками программирования, как C/C++, Python, Java и Matlab.

CoppeliaSim использует кинематический движок для прямых и обратных расчетов кинематики, а также несколько библиотек для моделирования физики (MuJoCo, Bullet, ODE, Vortex, Newton Game Dynamics) для моделирования твердого тела. Модели и сцены строятся путем сборки различных объектов (сетки, стыки, различные датчики, облака точек, октодеревья и т.д.) в иерархическую структуру. Дополнительные функциональные возможности, предоставляемые плагинами, включают: планирование движения (через OMPL), синтетическое зрение и обработку изображений (например, через OpenCV), обнаружение столкновений, расчет минимального расстояния, пользовательские графические интерфейсы и визуализацию данных (например, с помощью графиков).

Основными областями применения CoppeliaSim являются исследования в области робототехники и образование[20].

# **2.2 Локализация ориентиров в поле зрения с помощью фильтра Калмана.**

Решение задачи SLAM методами на основе расширенного фильтра Калмана являются наиболее распространенными. Фильтр Калмана является

рекурсивным фильтром, который оценивает вектор состояния динамической

системы, используя ряд зашумленных измерений. Алгоритм EKF-SLAM

позволяет не только уточнять оценку положения мобильной платформы на

карте, но и положение всех обнаруженных ориентиров. При всей своей

привлекательности, EKF тем не менее имеет свои недостатки, к которым

можно отнести в первую очередь ограничение на количество ориентиров в

системе.

Задача SLAM на базе расширенного фильтра Калмана состоит из

следующих основных частей:

1) Обнаружение ориентиров в пространстве.

2) Ассоциация данных, т. е. поиск соответствий ориентиров.

3) Оценка месторасположений мобильной платформы и ориентиров.

4) Уточнение месторасположений мобильной платформы и ориентиров.

Применяются различные способы реализации отдельных подзадач в

рамках указанной задачи. Таким образом, существует возможность

комбинировать различные реализации отдельных алгоритмов и улучшать их

по отдельности.

1. Обнаружение ориентиров в пространстве. Основная цель задачи

SLAM состоит в том, чтобы получить точную оценку месторасположения

мобильной платформы с помощью наблюдения и оценки месторасположения

ориентиров. Информация, необходимая для определения месторасположения

мобильной платформы, зависит от точности используемого сенсора и

точности оценок месторасположения наблюдаемых ориентиров.

Геометрическое распределение ориентиров также играет роль. Например,

когда нужно ограничить число ориентиров на карте, более выгодно

сохранять ориентиры, удаленные друг от друга, и чем те, которые близки

друг к другу. Кроме того, оценки ориентиров, которые плохо коррелированы,

предоставляют больше информации. Когда две оценки ориентиров

полностью коррелированы, то такие ориентиры не дают полезной

информации, за исключением геометрической. Еще важнее выбирать

ориентиры, таким образом, чтобы была максимальная точность и

производительность у алгоритма.

2. Ассоциация данных, т.е. поиск соответствий ориентиров.

Ориентиры, обнаруженные датчиками аппарата на каждом шаге, включают в

себя уже существующие ориентиры на карте, а также новые ориентиры. Если

не провести соответствие найденных ориентиров с уже существующими на

карте, то на нее будут добавлены ориентиры-дубликаты, что приведет к

неправильной работе алгоритма SLAM. В большинстве алгоритмов SLAM на базе расширенного фильтра Калмана используют ассоциации ближайшего соседа. В нем для ассоциации каждого наблюдаемого ориентира проводится сравнение расстояния до существующих ориентиров с заранее заданным порогом. Этот простой алгоритм не учитывает взаимосвязь между ориентирами, поэтому вероятность неправильной ассоциации у алгоритма высока.

3.Оценка месторасположения мобильной платформы и ориентиров. Состояние мобильной платформы в произвольный момент времени можно представить с помощью вектора оценки месторасположения 𝑋 и ковариационной матрицы 𝑃.

4. Уточнение месторасположения мобильной платформы и ориентиров. Во время этапа корректировки происходит уточнение месторасположений объектов на карте, используя рассогласования между предсказанными месторасположениями наблюдаемыми ориентирами и полученными месторасположениями ориентиров с помощью датчиков [21].

# **2.3 Локализация ориентиров в поле зрения с помощью фильтра частиц.**

Навигация внутри помещений позволяет нам предоставлять навигационные решения и организовывать навигацию внутри здания. Для реализации навигации необходимо создать правильную модель позиционирования. Алгоритм оценивает положение объекта и строит статистическое распределение потенциальных положений объекта. Одним из подходов, используемых для создания вероятностной модели позиционирования, является фильтр частиц.

Фильтр частиц генерирует так называемые частицы. Каждая частица представляет определенное состояние системы - положение и ориентацию. Во-первых, в самом начале навигации генерируются частицы с нормальным распределением, потому что нет никакой информации о позиционировании устройства. Координаты и ориентации устройства распределяются в соответствии с геометрией местоположения. Эта процедура называется выборкой.

Каждая частица имеет свой параметр – вес. Это число представляет собой вероятность нахождения системы в определенном состоянии. Чем больше вес, тем больше вероятность найти пользователя в том или ином месте. Затем во время навигации измерения с датчиков дают более точную информацию о положении смартфона. Вес частицы рассчитывается в соответствии с измерениями. Для различных типов протоколов распределения данных применяются различные модели правдоподобия с определенными распределениями весов. Частицы, которые соответствуют положениям, которые не наблюдались на этапе сбора данных, «умирают».

Процедура повторного отбора проб создает больше частиц вокруг точек, которые находятся ближе к раствору, поэтому количество частиц компенсируется новыми. Для других частиц корректируется вес и корректируется общее положение пользователя на карте. Фильтр частиц называется методом локализации по методу Монте-Карло. Это также называется байесовской проблемой фильтрации.

Первоначальная генерация частиц происходит в той области, где можно найти устройство. Так как карта состоит из различных преград, таких как стены и другие препятствия, пользователя нельзя нигде разместить. Также важным моментом является то, что навигация возможна только в те места на карте, которые находятся близко к передатчикам для обеспечения стабильного покрытия сигнала. Именно поэтому инициализация частиц происходит только после первого пакета накопленных сигналов.

Положение каждой частицы можно предсказать с помощью траекторной информации от датчика. Регулировка положения обеспечивает длину шага и направление движения. Приращение положения и угла зависит от дополнительного значения, которое генерируется случайным образом на основе возможной длины и направления шага. Эта неопределенность нужна потому, что необходимо учесть все возможные изменения траектории. Частицы с позициями, которые не помещаются в положение, определяемое геометрией, считаются мертвыми (имеют нулевой вес). Иногда точность измерений не может обеспечить достаточное количество частиц с действительными положениями. В этом случае веса должны быть инициализированы заново.

При достаточном накоплении измерений происходит коррекция.

# **Глава 3. Разработка виртуальной модели местности и автономного шагающего робота, реализация тестового движения робота по местности и навигации робота по камере.**

# **3.1 Модель платформы и окружающей среды**

Проанализировав ряд видеоматериалов, полученных с камер, установленных на шагающих платформах, были сформулированы следующие требования к модели:

* Камера должна быть установлена в передней части робота
* Модель должна быть голономной и совершать плоскопараллельное движение относительно поверхности, как и оригинал
* На видео с камеры должна присутствовать ритмичная тряска, вызванная шагами

Учитывая данные требования в рамках данной работы предлагается следующее решение – использовать колесную платформу с колесами в форме неправильных звездчатых многоугольников (Рис.3), расположенных в разных плоскостях, для имитации движения шагающей платформы

(Рис. 4).

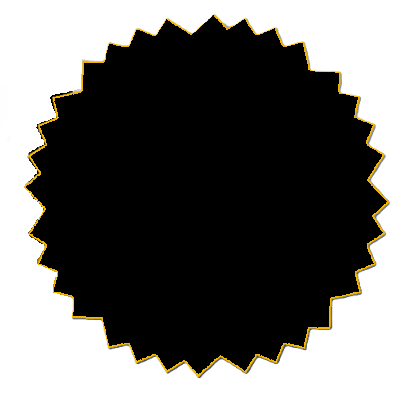


Рис.3 Профиль колеса в форме неправильного звездчатого многоугольника

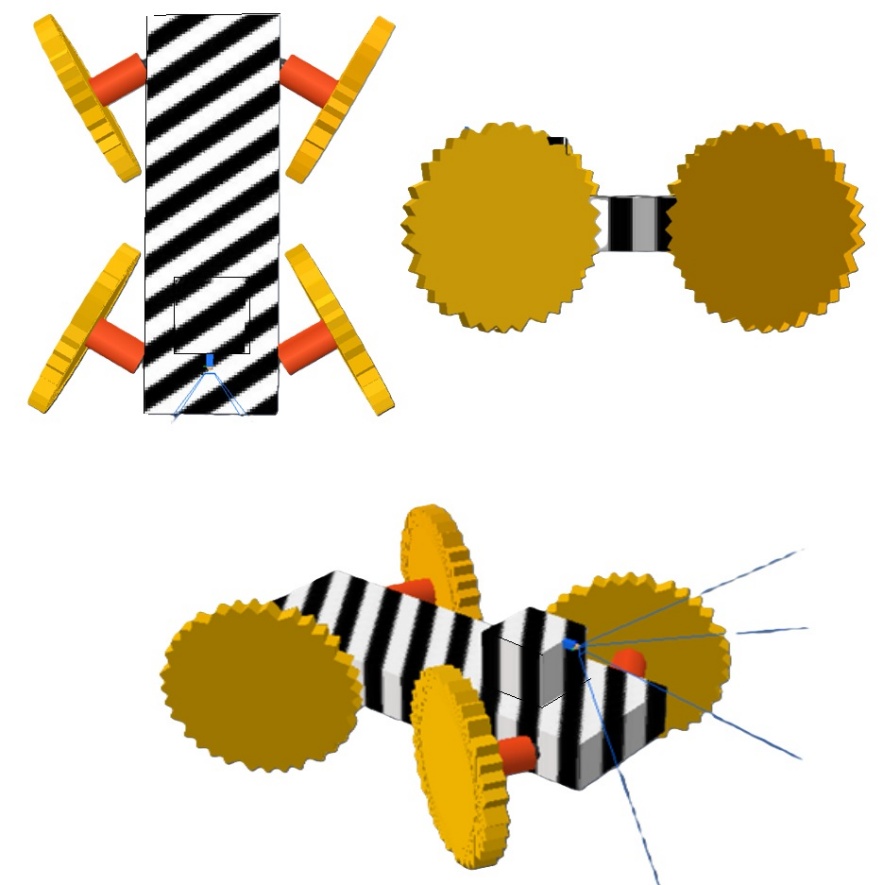
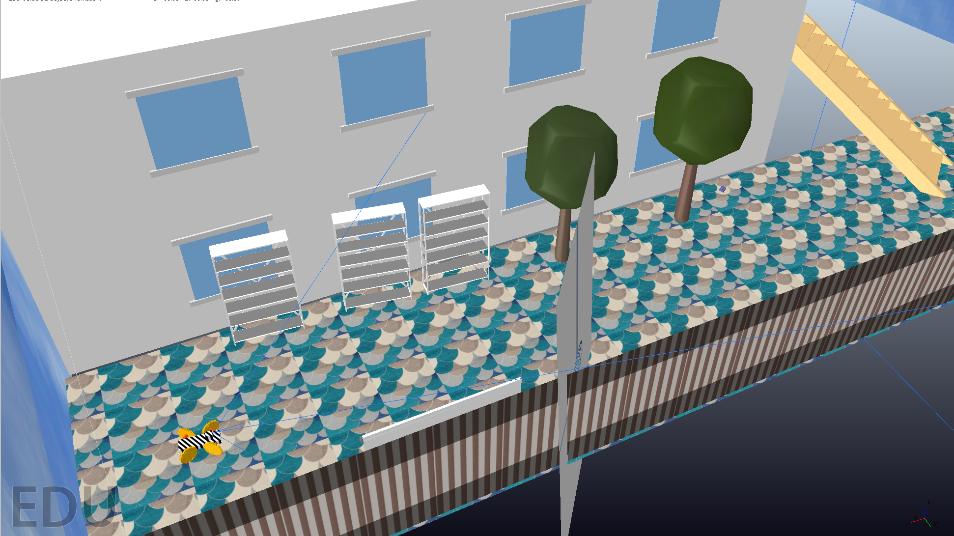
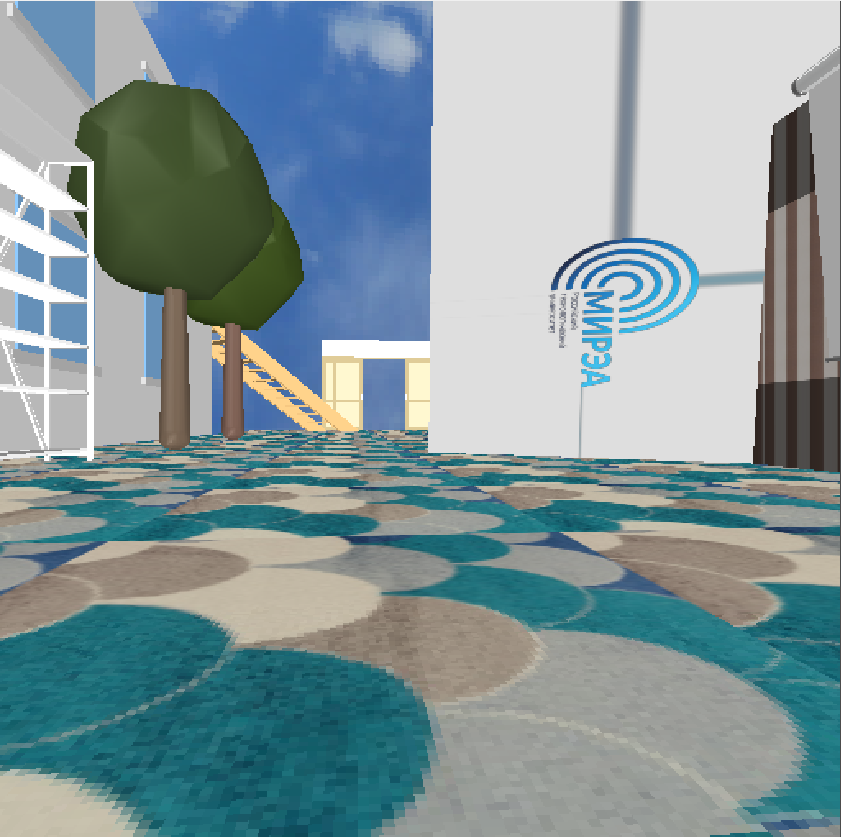


Рис. 4 Предлагаемая модель платформы для имитации движения шагающего робота

Модель содержит камеру, снимающую цветное видео разрешением 512х512 пикселей с частотой 20 кадров в секунду, поле зрения камеры 70°.

Для проведения исследования была построена модель среды, окружающей робота (Рис. 5). Она представляется прямоугольной локацией коридорного типа, все поверхности имеют различимые текстуры и контуры, ландшафт плоский, окружение статичное, освещение равномерное.

Рис. 5 Модель окружающей среды, робот в ней, вид с камеры робота в процессе симуляции видеопоток кадрируется и сохраняется в виде набора отдельных кадров в формате ‘jpg’, ниже (Рис 6) приведена примерная раскадровка тестового движения.

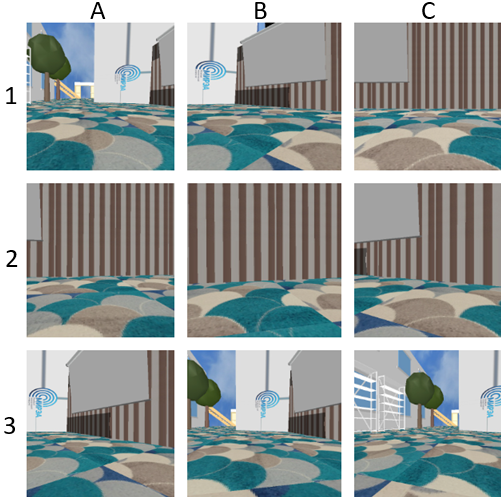


Рис. 6 Раскадровка тестового движения, с траекторией показанной на Рис 10 (справа), в период 4-12 секунды с дискретизацией 1 секунда

Скрипт ,реализующий движение робота, работает по принципу циклической ОСРВ, программа состоит из главного цикла с задаваемой частотой вызова и набора классов подпрограмм, унаследованных от абстрактного интерфейса, определяющего их минимальный набор «ручек», вызываемых в данном цикле. В рамках данной работы использовались такие подпрограммы как ограничитель времени, программа управления с клавиатуры, воспроизведение записанных инструкций, экстракторы данных из симуляции и загрузчики данных в файлы.

Код скрипта управления платформой и средой приведен в приложении.2

# **3.2 Восстановление траектории движения**

В рамках данной работы восстановление траектории движения происходит по следующему алгоритму:

* Извлечение особых точек из двух соседних кадров и вычисление их дескрипторов с помощью метода ORB (Oriented FAST & Rotated BRIEF)[3].
* Сопоставление дескрипторов и выделение наиболее точных признаков с помощью теста Лоу [4].
* Преобразование точек в нормализованное пространство устройства посредством умножения на обратную калибровочную матрицу камеры [5].
* Вычисление основной матрицы по особым точкам двух кадров с помощью метода RANSAC [6].
* Декомпозиция основной матрицы на матрицы поворота и смещения с помощью сингулярного разложения [7].
* Восстановление траектории сшиванием элементарных траекторий между каждой парой соседних кадров.

Реализация алгоритма показана в приложении 2 с.11-27

# **3.3 Применение метода ORB для поиска точек и вычисления дескрипторов**

В процессе разработки для извлечения особых точек и вычисления их дескрипторов использовался модуль ORBExtractor библиотеки python\_orb\_slam3 с открытым исходным кодом. Принцип алгоритма был изложен выше (см. главу 1).



Рис. 7 Особые точки и некоторые наиболее сильные совпадения на двух соседних кадрах обнаруженные модулем ORBExtractor

# **3.4 Сопоставление дескрипторов и выделение наиболее точных признаков с помощью теста Лоу**

Для ускорения последующих обращений все дескрипторы помещаются в Kd-дерево, где ключами являются векторы дескриптора, а хранимые значения – индексы данных точек. Из-за того, что изменения между кадрами незначительны, а вычисление дескриптора BRIEF учитывает всего 256 точек из окрестности особой точки, то распространенной проблемой является наличие нескольких особых точек с одинаковыми векторами дескриптора. Для определения истинной точки (что наиболее точным образом сопоставляется с истинной) применяется тест Лоу:

* Каждая ключевая точка первого изображения сопоставляется с потенциальными точками второго изображения, которые находятся как ближайшие соседи в дереве.
* Рассматриваются по два лучших соответствия для каждого уникального дескриптора (лучшими считаются соответствия с наименьшим расстоянием).
* Рассматриваются по два лучших соответствия для каждой точки (лучшими считаются соответствия с наименьшим расстоянием).

Для реализации концепции «достаточной разницы» используется соотношение двух расстояний. Например, если расстояние между ключевой точкой и её наилучшим совпадением меньше, чем расстояние между ключевой точкой и её вторым лучшим совпадением, но при этом отношение расстояний не превышает определённого порога, точка проходит тест. Обычно порог устанавливается примерно на 0.5 (в данной работе порог принят равным 0.8), что означает, что наилучшее соответствие должно быть как минимум в два раза ближе (в данной работе в 5 раз), чем второе наилучшее [4].

# **3.5 Преобразование точек в нормализованное пространство устройства посредством умножения на обратную калибровочную матрицу камеры**

В рамках данной работы используется виртуальная модель камеры, избавленная от артефактов и оптических искажений, также камера не нуждается в калибровке, ее калибровочная матрица известна изначально:

В реальном случае, таких «идеальных» камер не существует и требуется проведение калибровки камеры – то есть получения ее калибровочной матрицы. В самом распространенном случае, когда у пользователя есть доступ к камере это делается следующим образом: перед камерой устанавливается шаблон – простое контрастное изображение, как правило шахматные клетки, система координат камеры связывается с системой координат шаблона через распознавание углов клеток. По угловым точкам через решение системы уравнений определяются параметры калибровки [16].

# **3.6 Вычисление основной матрицы по особым точкам двух кадров**

Основная матрица связывает отношения точек между изображениями и определяет, какая точка на одном кадре соответствует какой на другом. Для точного расчёта важно исключить ложные точки, которые могут привести к ошибкам.

Метод RANSAC (Random sample consensus) позволяет определить набор подходящих точек. Это итеративный метод оценки параметров математической модели из набора наблюдаемых данных, содержащих выбросы , когда выбросам не должно быть предоставлено никакого влияния на значения оценок. Поэтому его также можно интерпретировать как метод обнаружения выбросов [17].

Применением данного алгоритма в этой работе является подгонка линий в двух измерениях к набору наблюдений (Рис. 8). На примере одной линии это происходит так: предполагается, что этот набор содержит как инлайеры , т. е. точки, которые приблизительно могут быть подогнаны к линии, так и выбросы , точки, которые не могут быть подогнаны к этой линии, простой метод наименьших квадратов для подгонки линии, как правило, даст линию с плохим соответствием данным, включая инлайеры и выбросы. Причина в том, что он оптимально подгоняется ко всем точкам, включая выбросы. RANSAC, с другой стороны, пытается исключить выбросы и найти линейную модель, которая использует только инлайеры в своих расчетах. Это делается путем подгонки линейных моделей к нескольким случайным выборкам данных и возврата модели, которая имеет наилучшее соответствие подмножеству данных. Поскольку инлайеры, как правило, более линейно связаны, чем случайная смесь инлайеров и выбросов, случайное подмножество, состоящее полностью из инлайеров, будет иметь наилучшее соответствие модели. На практике нет гарантии, что подмножество инлайеров будет выбрано случайным образом, а вероятность успеха алгоритма зависит от доли инлайеров в данных, а также от выбора нескольких параметров алгоритма [18].

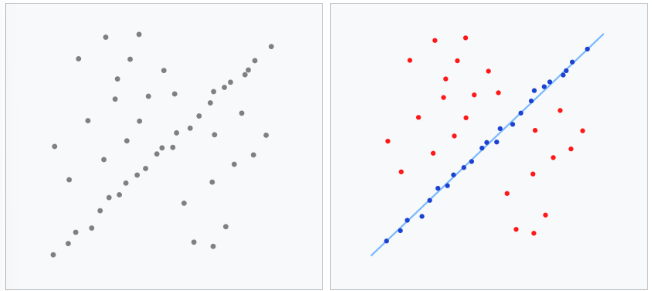


Рис. 8 Набор данных со множеством выбросов, для которых необходимо построить линию (слева) и подобранная линия с помощью RANSAC (справа); выбросы не влияют на результат.

Вернемся к нахождению основной матрицы. Алгоритм ее нахождения содержит следующие шаги:

* Формирование основного набора точек для вычисления матрицы. Для этого случайным образом выбираются несколько точек из множества соответствий, а остальные проверяются на условие: точки должны лежать на эпиполярных линиях.
* Отбрасывание ложных точек. Сопоставленные пары, в которых точка находится на расстоянии от её эпиполярной линии, превышающем пороговое значение, считаются ложными и отбрасываются.
* Повторение данного процесса несколько раз, с целью получения разных основных наборов точек.
* Вычисление основной матрицы на основе наибольшего основного набора точек, такая матрица наиболее точно соответствует действительности (Рис. 9) [14].

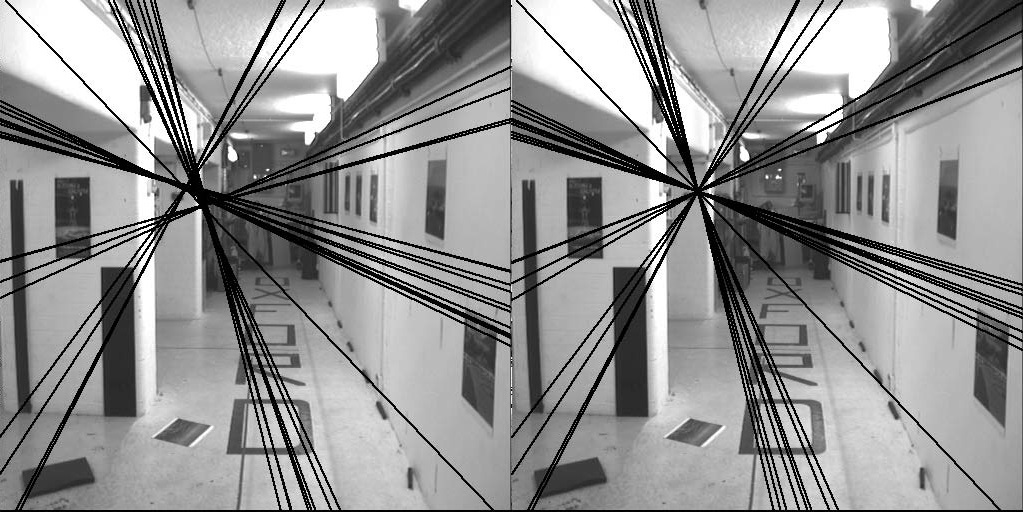


Рис.9. пример эпиполярных линий, полученных из правильной фундаментальной матрицы (картинка справа) и неправильной (слева) [14].

# **3.7 Декомпозиция основной матрицы**

Выполняя сингулярное разложение (SVD) основной матрицы, мы получаем три компонента: две ортогональные матрицы и диагональную матрицу с сингулярными значениями. Из этих компонент можно извлечь матрицу вращения R (первая ортогональная матрица) и вектор смещения t (значения главной диагонали второй ортогональной матрицы. (Напоминание о том как производится сингулярное разложение матрицы вынесено в прил.1)

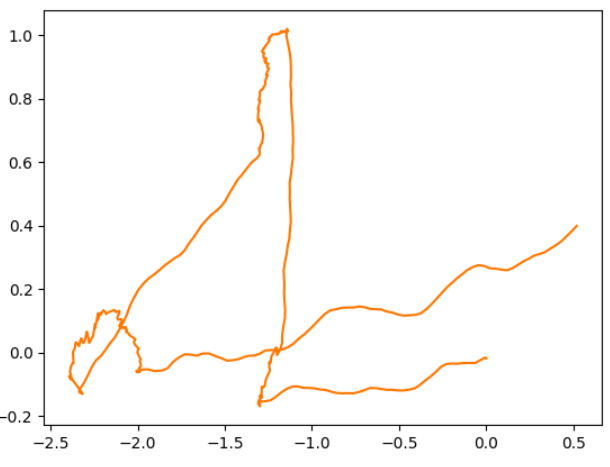
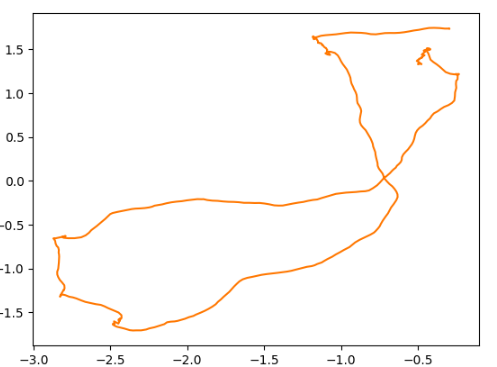
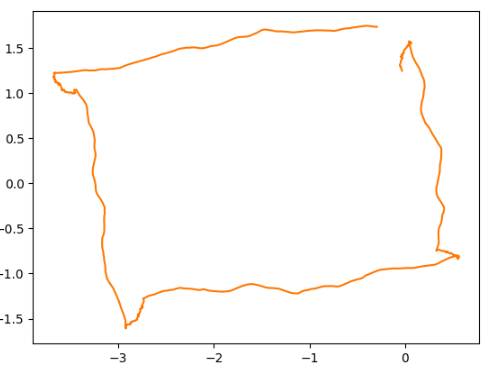
**3.8 Восстановление траектории**

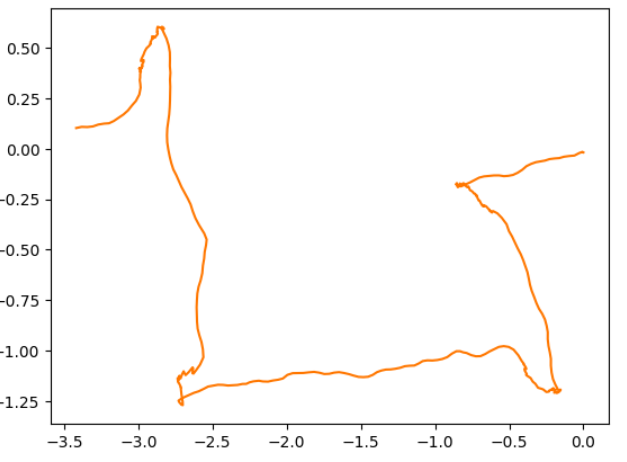
Имея после предыдущих шагов значения матриц поворота и векторов смещения между положениями камеры в моменты съемки каждого отельного кадра видеоряда, производится восстановление траектории посредством сращивания воедино всех векторов перемещений с учетом ориентации робота, ориентация робота в момент совершения каждого кадра определяется восстановлением угла тангажа из соответственного кватерниона по формуле:

Для кватерниона вида: ,

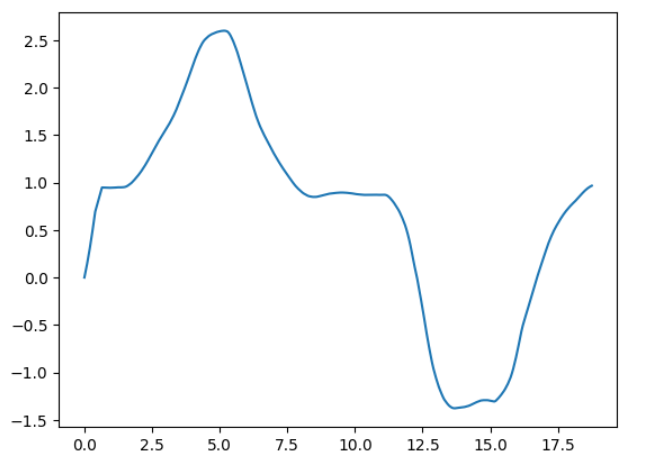
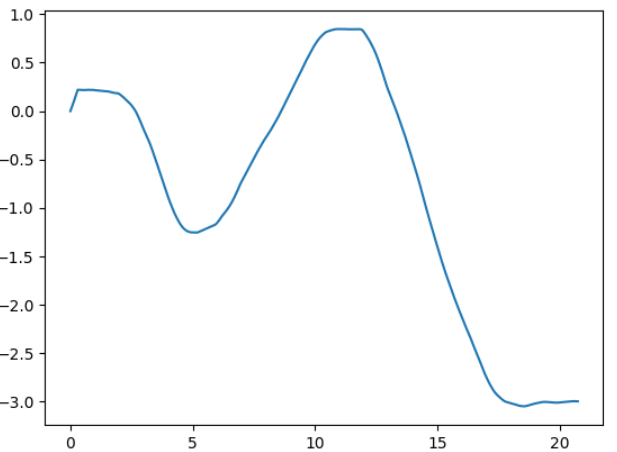
Соответствующие кватернионы в свою очередь определяются по второй теореме Пуассона для кинематики [19].

# **3.9 Результат восстановления траектории по видео полученному во время тестового движения**





В рамках эксперимента произведены тестовые движения по следующим траекториям (Рис.10), по результатам обработки измерений вышеописанными методами, получены следующие значения ориентаций вокруг вертикальной оси (Рис.11) и смещений робота в его собственной системе координат (Рис.12)

Рис.10 – Истинные траектории движения робота на плоскости

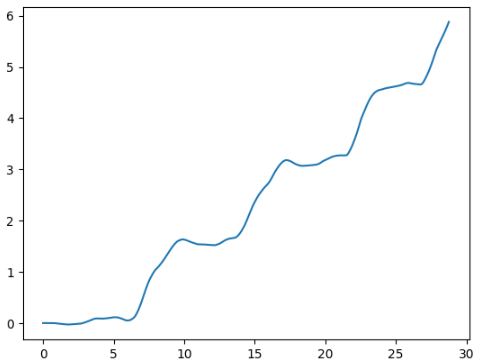
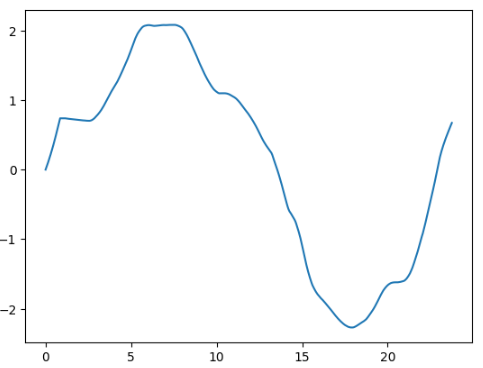
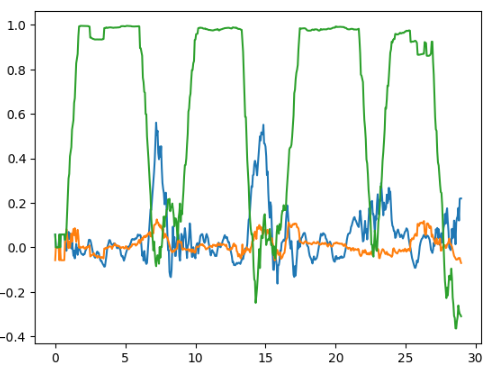
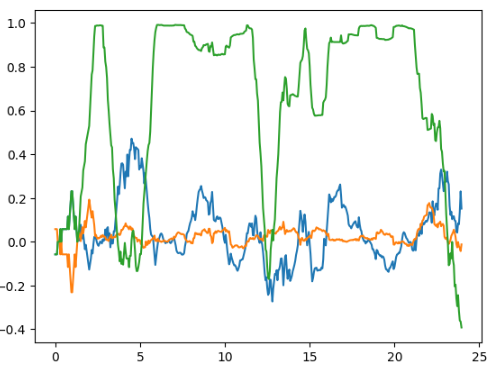
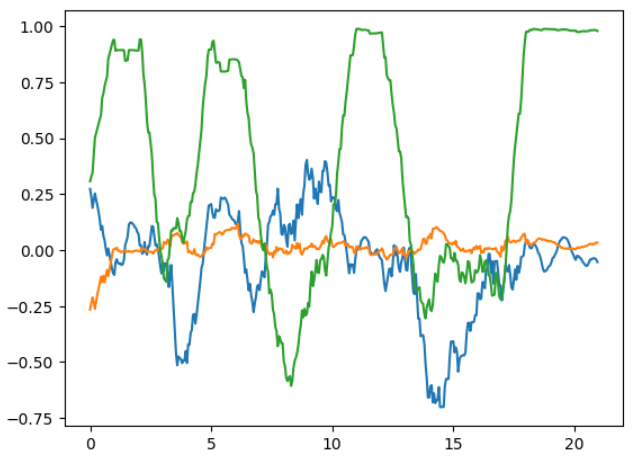
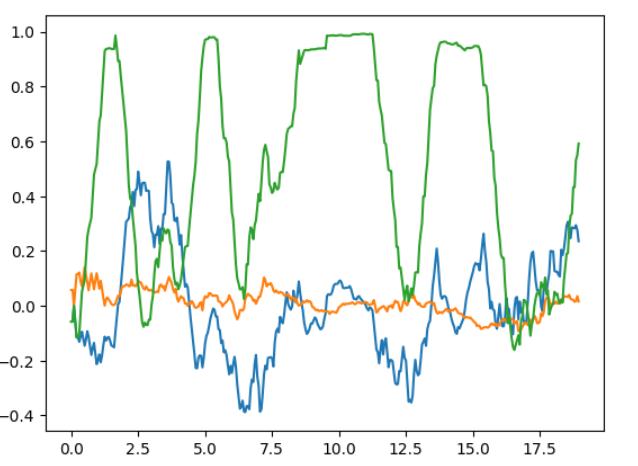
 (масштаб в метрах, движение из начала координат)

Рис.11 – Значения ориентации (рад) от времени (с) в процессе движения восстановленное с данных камеры

Рис.12 – Значения смещения камеры по осям x,y,z (синий, оранжевый, зеленый соответственно) от времени, восстановленные с данных камеры

Полагаясь на эти данные произведено восстановление траектории (Рис.14), получены сведения о нарастании ошибки на каждом шаге по следующей формуле:

где –вклад i-го восстановленного положения на итоговое значение ошибки позиционирования:

- радиус вектор, описывающий истинное положение робота на i-ом шаге

- радиус вектор, описывающий восстановленное положение робота на i-ом шаге.

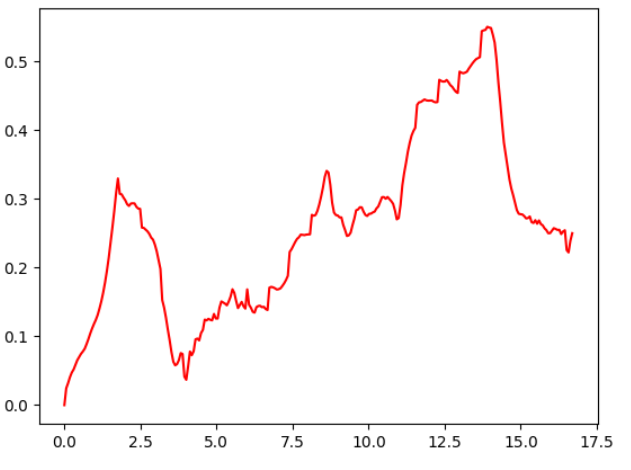
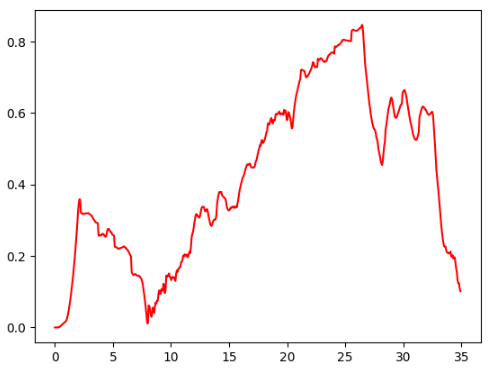
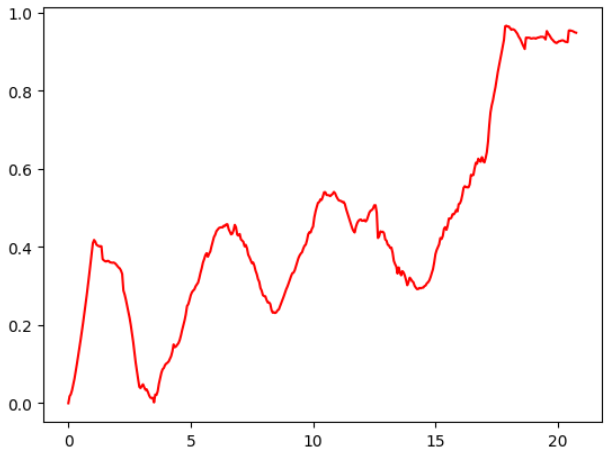
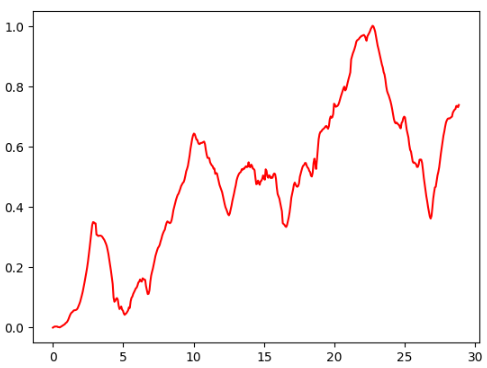
Соответственные графики, характеризующие абсолютное значение ошибки позиционирования для каждого момента восстановленной траектории приведены ниже:

Рис.13 Значения ошибки позиционирования от времени для тестовых движений

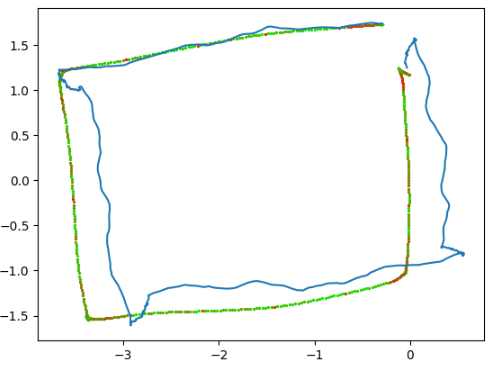
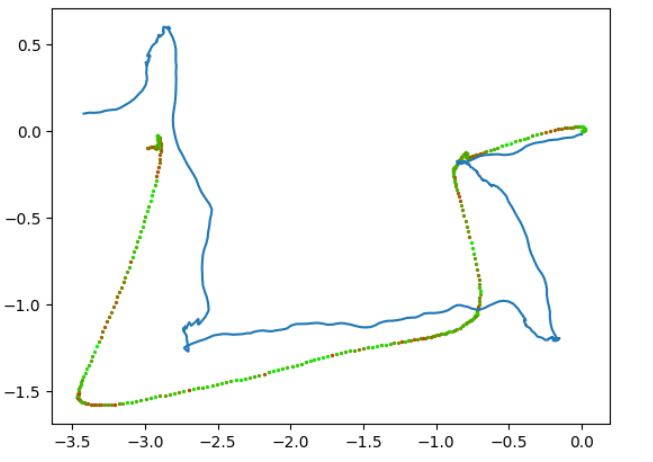
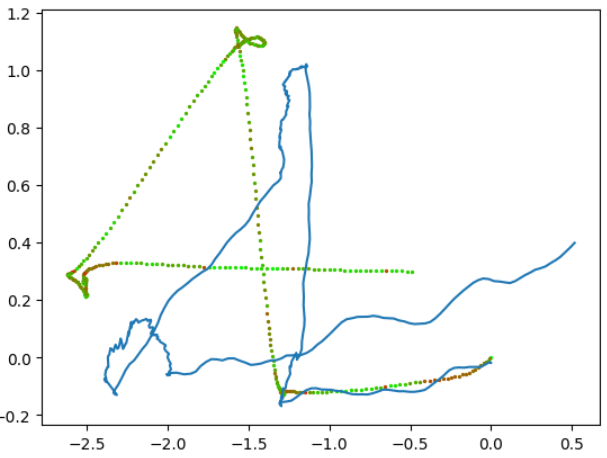
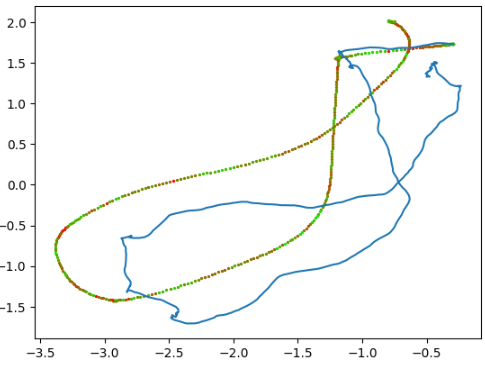


Рис.14 – траектория, восстановленная с данных камеры в сравнении с истинной траекторией (синим цветом), цветами на восстановленной траектории показано накопление ошибки в каждый их моментов времени, красный – ошибка нарастает, зеленый – нарастания ошибки нет

Скрипт восстановления траектории реализован по следующему принципу, видеопоток разбитый покадрово последовательно загружается в программу, где производятся вышеописанные вычисления. Результирующая траектория сопоставляется с истинной, вычисляются ошибки и их производные для анализа динамики накопления ошибки.

Реализация скрипта восстановления траектории изложена в приложении 2 с.113-133.

В результате анализа графиков сделан заключение, что средняя скорость накопления ошибки позиционирования при движении по прямой со средней линейной скоростью 1 м/с составила 1.47 см/сек.

Средняя скорость накопления ошибки позиционирования при движении по окружности радиуса 1м со средней линейной скоростью 1 м/с (угловая скорость примерно равно 1 рад/с) составила 10 см/сек.

# **Заключение**

По результатам проведенных тестов выявлена достаточно высокая точность визуальной навигации на прямых участках, однако также выявлено серьезное снижение точности при наличии угловой скорости. В результате чего возможно сделать выводы о необходимой частоте сверки с картой при применении SLAM в зависимости от характера совершаемого движения. Разработано программное обеспечение для более удобного взаимодействия со средой моделирования, что должно упростить последующую работу с разрабатываемыми и тестируемыми моделями.

## **Список использованной литературы**

1 - Hamsa Datta Peru Comparative analysis of ROS based 2D and 3D SLAM algorithms for Autonomous Ground Vehicles/ Conference: 2020 International Conference for Emerging Technology (INCET)

2 - Pedrosa, E., Reis L., Silva C.M.D., Ferreira H.S. Autonomous Navigation with Simultaneous Localization and Mapping in/outdoor. 2020

3-Rublee, Ethan; Rabaud, Vincent; Konolige, Kurt; Bradski, Gary (2011). "ORB: an efficient alternative to SIFT or SURF". IEEE International Conference on Computer Vision (ICCV).

4-David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints // International Journal of Computer Vision. — 2004. — Т. 60, вып. 2. — С. 91—110.

5- Пономарев В.В. Машинная графика. Учебное пособие. Ред. 2. Озерск: ОТИ МИФИ, 2006. – 72 с. ил.

6-Martin A. Fischler and Robert C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography (англ.) // Comm. Of the ACM : journal. — 1981. — June (vol. 24). — P. 381—395

7-Hartley, Richard; Andrew Zisserman (2004). Multiple view geometry in computer vision (2nd ed.). Cambridge, UK. ISBN 978-0-511-18711-7. OCLC 171123855

8 - Rodehorst V., Koschan A. / Comparison and evaluation of feature point detectors, 2006.

9 - Rosten, Edward; Drummond, Tom (2006). "Machine Learning for High-speed Corner Detection". Computer Vision – ECCV 2006. Lecture Notes in Computer Science. Vol. 3951. pp. 430–443.

10- Edward Rosten, Gerhard Reitmayr and Tom Drummond / Real-time Video Annotations for Augmented Reality / Department of Engineering, University of Cambridge, Cambridge CB1 2PZ, UK

11 - 2. Michael Calonder, Vincent Lepetit, Christoph Strecha, Pascal Fua / BRIEF: Binary Robust Independent Elementary Features // 11th European Conference on Computer Vision (ECCV), pp. 778 – 792, 2010.

12- Алпатов Б.А., Муравьев В.С, Стротов В.В, Фельдман А.Б/ Исследование эффективности применения алгоритмов анализа изображений в задаче навигации беспилотных летательных аппаратов/ Цифровая Обработка Сигналов №3/2012/ УДК 004.932

13- Платонов А. К. Определение параметров проективного отображения в зрительном канале робота // Препринты ИПМ им. М.В.Келдыша. 2007. № 39. 33 с.

14- Основы стереозрения // habr.com URL: https://habr.com/ru/articles/130300/ (дата обращения: 29.05.2025).

15- Hartley, R.~I. and Zisserman, A. Multiple View Geometry in Computer Vision. - 2 изд. - Cambridge: Cambridge University Press, ISBN: 0521540518, 2004. - 607 с.

16-Никитин, В. Н. Калибровка камер по снимкам плоского тест-объекта / В. Н. Никитин, А. В. Семенцов // Изв. вузов. Геодезия и аэрофотосъемка. – 2014. – № 2.

17- Подбор данных и неопределенность, Т. Струц, Springer Vieweg (2-е издание, 2016 г.)

18- Random sample consensus // wikipedia URL: https://en.wikipedia.org/wiki/Random\_sample\_consensus (дата обращения: 30.05.2025).

19- B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. Journal of Optical Society of America A, 4(4) :629–642,

20- CoppeliaSim // wikipedia URL: https://en.wikipedia.org/wiki/CoppeliaSim?ysclid=mbgun5we3n206292434 (дата обращения: 03.06.2025).

21- Антипов В.А. Повышение точности позицонирования камеры в системе прикладного телевидения с использованием расширенного фильтра Калмана : дис. к.т.н.: 02.02.13. - Ярославль, 2021. - 118 с.

# **Приложение 1**

**Сингулярное разложение:**

1. Для произвольной матрицы A размера m×n сингулярное разложение строится следующим образом:
2. Вычислить матрицу AᵀA размера n×n.
3. Найти собственные значения и собственные векторы матрицы AᵀA.
4. Нормализовать каждый из векторов так, чтобы его длина была равна 1.
5. Найти сингулярные числа матрицы A как корни из найденных собственных значений матрицы AᵀA: σi = √λi, где i = 1,…,k, где k = min(m, n).
6. Сформировать диагональную матрицу Σ размера m×n, состоящую из найденных сингулярных значений.
7. Построить матрицы U и V из собственных векторов AᵀA и AAᵀ соответственно.

**Иерархия классов входящих в состав модели робота в среде моделирования:**

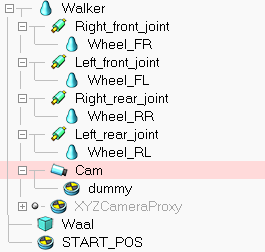


Рис. П.1 – Иерархия классов робота Walker в среде CoppeliaSim

# **Приложение 2**

Код скрипта управляющего моделью робота и среды

1. from coppeliasim\_zmqremoteapi\_client import RemoteAPIClient
2. import cv2
3. import numpy as np
4. import matplotlib.pyplot as plt
5. import keyboard
6. from abc import ABC, abstractmethod
7. import os
8. import csv
9. class Default\_robot\_interface(ABC):
10. def \_\_init\_\_(self, paths = None, simulation = None):
11. self.sim = simulation
12. self.linear\_velocity = 0
13. self.rotational\_velocity = 0
14. self.sidewalk\_velocity = 0
15. @abstractmethod
16. def UPDATE\_VEL(self):
17. pass
18. def Set\_linear\_velocity(self, value):
19. self.linear\_velocity = value
20. self.UPDATE\_VEL()
21. def Set\_rotational\_velocity(self, value):
22. self.rotational\_velocity = value
23. self.UPDATE\_VEL()
24. def Set\_sidewalk\_velocity(self, value):
25. self.sidewalk\_velocity = value
26. self.UPDATE\_VEL()
27. @abstractmethod
28. def Get\_camera\_image(self):
29. pass
30. @abstractmethod
31. def Get\_camera\_real\_cords(self):
32. pass
33. class Walker(Default\_robot\_interface):
34. def \_\_init\_\_(self, paths = None, simulation = None):
35. super().\_\_init\_\_(paths, simulation)
36. self.Right\_front\_joint = paths[0]
37. self.Right\_rear\_joint = paths[1]
38. self.Left\_front\_joint = paths[2]
39. self.Left\_rear\_joint = paths[3]
40. self.Cam = paths[4]
41. self.Dummy = paths[5]
42. def UPDATE\_VEL(self):
43. self.sim.setJointTargetVelocity(self.Right\_front\_joint,
    * + - 1. self.linear\_velocity - self.rotational\_velocity + self.sidewalk\_velocity)
44. self.sim.setJointTargetVelocity(self.Right\_rear\_joint,
    * + - 1. self.linear\_velocity - self.rotational\_velocity - self.sidewalk\_velocity)
45. self.sim.setJointTargetVelocity(self.Left\_front\_joint,
    * + - 1. self.linear\_velocity + self.rotational\_velocity - self.sidewalk\_velocity)
46. self.sim.setJointTargetVelocity(self.Left\_rear\_joint,
    * + - 1. self.linear\_velocity + self.rotational\_velocity + self.sidewalk\_velocity)
47. def Get\_camera\_image(self):
48. img, resX, resY = sim.getVisionSensorCharImage(self.Cam)
49. img = np.frombuffer(img, dtype=np.uint8).reshape(resY, resX, 3)
50. img = cv2.flip(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB), 0)
51. return img
52. def Get\_camera\_real\_cords(self, REL\_POINT= None):
53. if not REL\_POINT:
    1. REL\_POINT = self.sim.getObject("./Floor")
54. pos = self.sim.getObjectPosition(self.Dummy, REL\_POINT)
55. return pos
56. class Simulation\_loop\_handler:
57. def \_\_init\_\_(self, Default\_robot\_interface, LOOP\_NODE\_list, simulation):
58. self.Robot = Default\_robot\_interface
59. self.LOOP\_NODE\_list = LOOP\_NODE\_list
60. self.loop\_memory\_dict = dict()
61. self.sim = simulation
62. self.running = False
63. def start(self):
64. self.sim.startSimulation()
65. self.running = True
66. while self.running:
    1. for NODE in self.LOOP\_NODE\_list:
       1. if possible\_returnment := NODE.call(self):
       2. self.loop\_memory\_dict[possible\_returnment[0]] = possible\_returnment[1]
    2. self.sim.step()
    3. print(f'Simulation time: {self.sim.getSimulationTime():.2f} [s]',sep='', end='\r')
67. self.sim.stopSimulation()
68. self.loop\_memory\_dict.clear()
69. class NODE\_interface(ABC):
70. @abstractmethod
71. def \_\_init\_\_(self, params = None):
72. pass
73. @abstractmethod
74. def call(self, Simulation\_loop\_handler, values = None):
75. pass
76. class SIM\_TIME\_LIMIT\_CHECKER(NODE\_interface):
77. def \_\_init\_\_(self, params = None):
78. self.time\_limit = params[0]
79. def call(self, Simulation\_loop\_handler):
80. if Simulation\_loop\_handler.sim.getSimulationTime() >= self.time\_limit:
    1. Simulation\_loop\_handler.running = False
81. class KEYBOARD\_REMOTE\_CONTROLLER(NODE\_interface):
82. def \_\_init\_\_(self, params = None):
83. self.velocity\_multiplier = params[0]
84. def call(self, Simulation\_loop\_handler):
85. if keyboard.is\_pressed('w'):
    1. Simulation\_loop\_handler.Robot.Set\_linear\_velocity(self.velocity\_multiplier)
86. elif keyboard.is\_pressed('s'):
    1. Simulation\_loop\_handler.Robot.Set\_linear\_velocity(-self.velocity\_multiplier)
87. else:
    1. Simulation\_loop\_handler.Robot.Set\_linear\_velocity(0)
88. if keyboard.is\_pressed('a'):
    1. Simulation\_loop\_handler.Robot.Set\_sidewalk\_velocity(-self.velocity\_multiplier)
89. elif keyboard.is\_pressed('d'):
    1. Simulation\_loop\_handler.Robot.Set\_sidewalk\_velocity(self.velocity\_multiplier)
90. else:
    1. Simulation\_loop\_handler.Robot.Set\_sidewalk\_velocity(0)
91. if keyboard.is\_pressed('q'):
    1. Simulation\_loop\_handler.Robot.Set\_rotational\_velocity(self.velocity\_multiplier)
92. elif keyboard.is\_pressed('e'):
    1. Simulation\_loop\_handler.Robot.Set\_rotational\_velocity(-self.velocity\_multiplier)
93. else:
    1. Simulation\_loop\_handler.Robot.Set\_rotational\_velocity(0)
94. class IMAGE\_LOADER(NODE\_interface):
95. def \_\_init\_\_(self, params = None):
96. self.load\_dir = params[0]
97. if self.load\_dir:
    1. for filename in os.listdir(self.load\_dir):
       1. file\_path = os.path.join(self.load\_dir, filename)
       2. try:
       3. if os.path.isfile(file\_path):
       4. os.remove(file\_path)
       5. except Exception as e:
       6. print(f'Ошибка при удалении файла {file\_path}. {e}')
98. def call(self, Simulation\_loop\_handler):
99. im = Simulation\_loop\_handler.Robot.Get\_camera\_image()
100. Simulation\_loop\_handler.loop\_memory\_dict['last\_gotten\_image'] = im
101. if self.load\_dir:
102. cv2.imwrite(f"{self.load\_dir}/{round(Simulation\_loop\_handler.sim.getSimulationTime(), 2)}.jpg", im)
103. class TRUE\_POS\_GETTER(NODE\_interface):
104. def \_\_init\_\_(self, params = None):
105. self.csv\_path = params[0]
106. self.relative\_point = params[1]
107. with open(self.csv\_path, 'w', newline='') as file:
     1. writer = csv.writer(file)
     2. writer.writerow([])
108. def call(self, Simulation\_loop\_handler):
109. pos = Simulation\_loop\_handler.Robot.Get\_camera\_real\_cords(self.relative\_point)
110. Simulation\_loop\_handler.loop\_memory\_dict['current\_position'] = pos
111. with open(self.csv\_path, 'a', newline='') as file:
     1. writer = csv.writer(file)
     2. pos.insert(0, round(Simulation\_loop\_handler.sim.getSimulationTime(), 2))
     3. writer.writerows([pos])
112. class IMAGE\_SHOWER(NODE\_interface):
113. def \_\_init\_\_(self, params = None):
114. pass
115. def call(self, Simulation\_loop\_handler):
116. plt.imshow(Simulation\_loop\_handler.loop\_memory\_dict['last\_gotten\_image'])
117. class Motion\_strategy\_in\_time:
118. def \_\_init\_\_(self, time2command\_list):
119. self.time2command\_list = time2command\_list
120. self.\_\_current\_instruction\_number = 0
121. if self.time2command\_list[0][0]:
     1. self.time2command\_list.insert(0, [0.0, [0, 0, 0]])
122. def get\_instruction(self, time):
123. if (self.\_\_current\_instruction\_number + 1 < len(self.time2command\_list) and
     1. time >= self.time2command\_list[self.\_\_current\_instruction\_number + 1][0]):
     2. self.\_\_current\_instruction\_number += 1
124. return self.time2command\_list[self.\_\_current\_instruction\_number][1]
125. class PLAN\_CONTROLLER(NODE\_interface):
126. def \_\_init\_\_(self, params = None):
127. self.strategy = params[0]
128. def call(self, Simulation\_loop\_handler):
129. velocities = self.strategy.get\_instruction(Simulation\_loop\_handler.sim.getSimulationTime())
130. Simulation\_loop\_handler.Robot.Set\_linear\_velocity(velocities[0])
131. Simulation\_loop\_handler.Robot.Set\_sidewalk\_velocity(velocities[1])
132. Simulation\_loop\_handler.Robot.Set\_rotational\_velocity(velocities[2])
133. client = RemoteAPIClient()
134. sim = client.require('sim')
135. client.setStepping(True)
136. walker = Walker(paths = [sim.getObject("./Walker/Right\_front\_joint"),
     * 1. sim.getObject("./Walker/Right\_rear\_joint"),
       2. sim.getObject("./Walker/Left\_front\_joint"),
       3. sim.getObject("./Walker/Left\_rear\_joint"),
       4. sim.getObject("./Walker/Cam"),
       5. sim.getObject("./Walker/Cam/dummy")],
       6. simulation = sim)
137. t\_lim = SIM\_TIME\_LIMIT\_CHECKER([20])
138. k\_rem = KEYBOARD\_REMOTE\_CONTROLLER([5])
139. plan = Motion\_strategy\_in\_time([[1,[5,0,0]],

[5, [0, 0, -5]],

[7, [-3, 0, 0]],

[11, [0, 0, -5]],

[13, [5, 0, 0]]])

1. RC = PLAN\_CONTROLLER([plan])
2. f\_loader = IMAGE\_LOADER(["./images\_from\_cam"])
3. #ish = IMAGE\_SHOWER()
4. tpg = TRUE\_POS\_GETTER(['True\_path.csv', sim.getObject("./START\_POS")])
5. NODES = [t\_lim,
6. f\_loader,
7. RC,
8. tpg]
9. MAIN\_SIM = Simulation\_loop\_handler(Default\_robot\_interface = walker,

LOOP\_NODE\_list = NODES,

simulation = sim)

1. MAIN\_SIM.start()

Код скрипта производящего расчет траектории

* + - 1. import cv2
      2. import os
      3. import matplotlib.pyplot as plt
      4. from python\_orb\_slam3 import ORBExtractor
      5. import numpy as np
      6. from scipy.spatial import cKDTree
      7. import random
      8. import pandas as pd
      9. import quaternion
      10. import copy
      11. def estimate\_camera\_motion(keypoints1, keypoints2, descriptors1, descriptors2, K):
      12. # Создаем дерево кД для быстрого поиска ближайших соседей
      13. tree = cKDTree(descriptors2)
      14. distances, indices = tree.query(descriptors1, k=2)
      15. # Применяем тест Лоу для фильтрации соответствий
      16. good\_matches = []
      17. for idx, (dist1, dist2) in enumerate(zip(distances[:, 0], distances[:, 1])):
      18. if dist1 < 0.8 \* dist2:
      19. good\_matches.append((keypoints1[idx], keypoints2[indices[idx][0]]))
      20. # Преобразуем соответствия в массивы координат
      21. src\_pts = np.float32([kp1.pt for kp1, \_ in good\_matches])
      22. dst\_pts = np.float32([kp2.pt for \_, kp2 in good\_matches])
      23. # Преобразуем точки в нормализованное пространство камеры
      24. src\_pts\_norm = cv2.undistortPoints(src\_pts.reshape(-1, 1, 2), K, None, P=K).reshape(-1, 2)
      25. dst\_pts\_norm = cv2.undistortPoints(dst\_pts.reshape(-1, 1, 2), K, None, P=K).reshape(-1, 2)
      26. # Вычисляем эссенциальную матрицу
      27. E, \_ = cv2.findEssentialMat(src\_pts\_norm, dst\_pts\_norm, K, method=cv2.LMEDS, prob=0.99, threshold=1.0)
      28. # Декомпозируем эссенциальную матрицу для получения R и t
      29. points, R, t, mask = cv2.recoverPose(E, src\_pts\_norm, dst\_pts\_norm, K)
      30. return R, t
      31. f = os.listdir('images\_from\_cam')
      32. f\_ext = [[float(x[:-4]), x] for x in f]
      33. f\_ext.sort()
      34. f = [x[1] for x in f\_ext]
      35. old\_frame = cv2.imread('images\_from\_cam/0.05.jpg')
      36. old\_gray = cv2.cvtColor(old\_frame, cv2.COLOR\_BGR2GRAY)
      37. orb\_extractor = ORBExtractor()
      38. old\_kp, old\_desc = orb\_extractor.detectAndCompute(old\_gray)
      39. focal\_length = old\_gray.shape[1]
      40. K = np.array([[focal\_length, 0, focal\_length/2],
      41. [0, focal\_length, focal\_length/2],
      42. [0, 0, 1]])
      43. Rs = []
      44. Ts = []
      45. for filename in f[2:]:
      47. frame = cv2.imread(os.path.join('images\_from\_cam',filename))
      48. frame\_gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)
      50. new\_kp, new\_desc = orb\_extractor.detectAndCompute(frame\_gray)
      51. R,T = estimate\_camera\_motion(old\_kp, new\_kp, old\_desc, new\_desc, K)
      52. old\_kp, old\_desc = new\_kp, new\_desc
      54. Rs.append(R)
      55. Ts.append(T)
      56. def moving\_average\_cumsum(data, window\_size):
      58. cumsum\_vec = np.cumsum(np.insert(data, 0, 0))
      59. return (cumsum\_vec[window\_size:] - cumsum\_vec[:-window\_size]) / window\_size
      60. offset = 4
      61. plt.plot(range(len(Rs[offset:])), [20\*x[0][1] for x in Rs[offset:]])
      62. plt.plot(range(len(Rs[offset:])), [20\*x[0][2] for x in Rs[offset:]])
      63. plt.plot(range(len(Rs[offset:])), [20\*x[1][2] for x in Rs[offset:]])
      64. w= 20
      65. fps= 20
      66. X\_w, Y\_w, Z\_w = (moving\_average\_cumsum([fps\*x[0][1] for x in Rs[offset:]],w),
      67. moving\_average\_cumsum([fps\*x[0][2] for x in Rs[offset:]],w),
      68. moving\_average\_cumsum([fps\*x[1][2] for x in Rs[offset:]],w))
      69. plt.plot([x/fps for x in range(len(X\_w))], X\_w)
      70. plt.plot([x/fps for x in range(len(Y\_w))], Y\_w)
      71. plt.plot([x/fps for x in range(len(Z\_w))], Z\_w)
      72. def yawFromQuat(quat):
      73. '''extract yaw angle from quaternion'''
      74. q0, q1, q2, q3 = quat
      75. return np.arctan2(2\*(q0\*q3 + q1\*q2), 1-2\*(q2\*\*2+q3\*\*2))
      76. def quatFromYaw(yaw):
      77. '''make quaternion from yaw angle'''
      78. q = np.array([np.cos(yaw/2), 0, 0, np.sin(yaw/2)])
      79. return q
      80. def smoothTh(th, thPrev):
      81. while th - thPrev > np.pi:
      82. th = th - 2\*np.pi
      83. while th - thPrev < -np.pi:
      84. th = th + 2\*np.pi
      85. return th
      86. def hamProd(a, b):
      87. '''Hamilton product of two quaternions'''
      88. return np.array([a[0]\*b[0]-a[1]\*b[1]-a[2]\*b[2]-a[3]\*b[3],
      89. a[0]\*b[1]+a[1]\*b[0]+a[2]\*b[3]-a[3]\*b[2],
      90. a[0]\*b[2]-a[1]\*b[3]+a[2]\*b[0]+a[3]\*b[1],
      91. a[0]\*b[3]+a[1]\*b[2]-a[2]\*b[1]+a[3]\*b[0]])
      93. def reconstruct\_with\_quaternion(X\_w, Y\_w, Z\_w, start\_angle=0, fps=20):
      94. qPrev = quatFromYaw(start\_angle)
      95. thPrev = start\_angle
      96. ans\_q = [qPrev]
      97. ans\_ang = [thPrev]
      98. for i in range(len(X\_w) - 1):
      99. qw = [0, X\_w[i], Z\_w[i], Y\_w[i]]
      100. qdot = 1/2 \* hamProd(qPrev, qw)
      101. dt = 1/fps
      102. q = qPrev + qdot\*dt
      103. ans\_q.append(q)
      104. th = smoothTh(yawFromQuat(q), thPrev)
      105. ans\_ang.append(th)
      106. qPrev = q
      107. thPrev = th
      108. return ans\_ang
      109. Y\_a = reconstruct\_with\_quaternion(X\_w, Y\_w, Z\_w,start\_angle=0)
      110. plt.plot([x/fps for x in range(len(Y\_a))], Y\_a)
      111. trans\_x = [-x[0][0] for x in Ts]
      112. trans\_y = [-x[1][0] for x in Ts]
      113. trans\_z = [-x[2][0] for x in Ts]
      114. ws= 20
      115. trans\_x = moving\_average\_cumsum(trans\_x, ws)
      116. trans\_y = moving\_average\_cumsum(trans\_y, ws)
      117. trans\_z = moving\_average\_cumsum(trans\_z, ws)
      118. plt.plot([x/fps for x in range(len(trans\_x))], trans\_x)
      119. plt.plot([x/fps for x in range(len(trans\_y))], trans\_y)
      120. plt.plot([x/fps for x in range(len(trans\_z))], trans\_z)
      121. df = pd.read\_csv('True\_path.csv')
      122. rx = [x[1] for x in df.to\_numpy()]
      123. ry = [x[2] for x in df.to\_numpy()]
      124. plt.plot(rx, ry,c='#ff7700')
      125. cordx = [0]
      126. cordy = [0]
      127. k =30.5
      128. D = -0.14
      129. for i in range(1,len(Y\_a)):
      130. cordx.append((trans\_y[i] \* np.sin(Y\_a[i] + D) + trans\_z[i] \* np.cos(Y\_a[i] + D)) + cordx[-1])
      131. cordy.append((trans\_y[i] \* np.cos(Y\_a[i] + D) + trans\_z[i] \* -np.sin(Y\_a[i] + D)) + cordy[-1])
      132. reference\_cords = [[x for x in rx], [x for x in ry]]
      133. decomposed\_cords = [[-x/k for x in cordx], [x/k for x in cordy]]
      134. plt.plot(decomposed\_cords[0], decomposed\_cords[1])
      135. plt.plot(reference\_cords[0], reference\_cords[1])
      136. shortcut\_idx= 0
      137. L2\_error = [0]
      138. sci\_step = len(reference\_cords[0])/len(decomposed\_cords[0])
      139. for i in range(1,len(decomposed\_cords[0])):
      140. nearest\_sci\_n = int(round(shortcut\_idx, 0))
      141. L2\_error.append(((reference\_cords[0][nearest\_sci\_n] - decomposed\_cords[0][i])\*\*2 +
      142. (reference\_cords[1][nearest\_sci\_n] - decomposed\_cords[1][i])\*\*2)\*\*0.5)
      143. shortcut\_idx += sci\_step
      144. plt.plot([x/fps for x in range(len(L2\_error))], L2\_error,c='#ff0000')
      145. L2\_error\_der = [L2\_error[i] - L2\_error[i-1] for i in range(1,len(L2\_error))]
      146. L2\_error\_der.insert(0,0)
      147. plt.plot([x for x in range(len(L2\_error))], L2\_error\_der,c='#9900ff')
      148. plt.plot(reference\_cords[0], reference\_cords[1])
      149. for i in range(len(decomposed\_cords[0])-1):
      150. plt.scatter(decomposed\_cords[0][i], decomposed\_cords[1][i], s=2,
      151. c=[[abs(L2\_error\_der[i]/max([abs(x) for x in L2\_error\_der]))\*\*0.45,1-abs(L2\_error\_der[i]/max([abs(x) for x in L2\_error\_der]))\*\*0.45,0]])