Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Факультет информационных технологий и программирования

Кафедра компьютерных технологий

**Д. Е. Родиков**

# Синхронное определение местоположения и составление 2D-карты по стерео изображению в режиме реального времени

## Бакалаврская работа

Научный руководитель – докт. техн. наук, профессор А.А. Шалыто

Санкт-Петербург

2010

## ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

Возможность навигации в реальной среде – базовое требование к автономному мобильному роботу. Роботу требуется отвечать на три вопроса: «Где я?», «Куда я хочу идти?», «Как туда попасть?» [1]. Для ответа на эти вопросы, необходимо знать к какой координатной системе и какому окружающему пространству они относятся. Система координат и информация о положении предметов в ней называется картой. Таким образом, для ответов на эти вопросы роботу требуется знать карту.

Реализация навигационной системы, которой заранее известны ориентиры или карта пространства, при наличии достаточно точных сенсоров, не представляет больших сложностей. Построение карты при постоянно известном положении робота также в значительной степени решенная проблема. Гораздо сложнее решать эти проблемы одновременно, не зная заранее ни карты, ни положения робота.

*Simultaneous localization and mapping* (*SLAM*) [2] – синхронное определение местоположения и составление карты. Задача *SLAM* связана с построением карты неизвестного пространства мобильным роботом во время навигации по строящейся карте.

Раньше решения данной задачи были трудоемкими и дорогими, использовались специальные методы: точное машиностроение, высокоточные сенсоры, калибровочные точки. Альтернативой является применение нескольких перекрывающихся сенсоров меньшего разрешения и совмещение пространственной информации от датчиков для получения более точных результатов [3]. Например, возможно совместное использование лазерного дальномера, видео камеры и информации о повороте и скорости вращения колес для автономного робота.

Задача *SLAM* состоит из множества частей [2]: нахождение ориентиров в пространстве, поиск их соответствий, вычисление местоположения, уточнение местоположения и положений ориентиров. Существуют различные способы реализации разных подзадач. Таким образом, появляется возможность комбинировать раз-

личные реализации отдельных алгоритмов и улучшать их по отдельности. Например, замена может потребоваться для работы на больших пространствах вместо работы в помещениях.

*SLAM* может применяться как для *2D*-, так и для *3D*-перемещений.

# 1. ЗАДАЧА *SLAM*

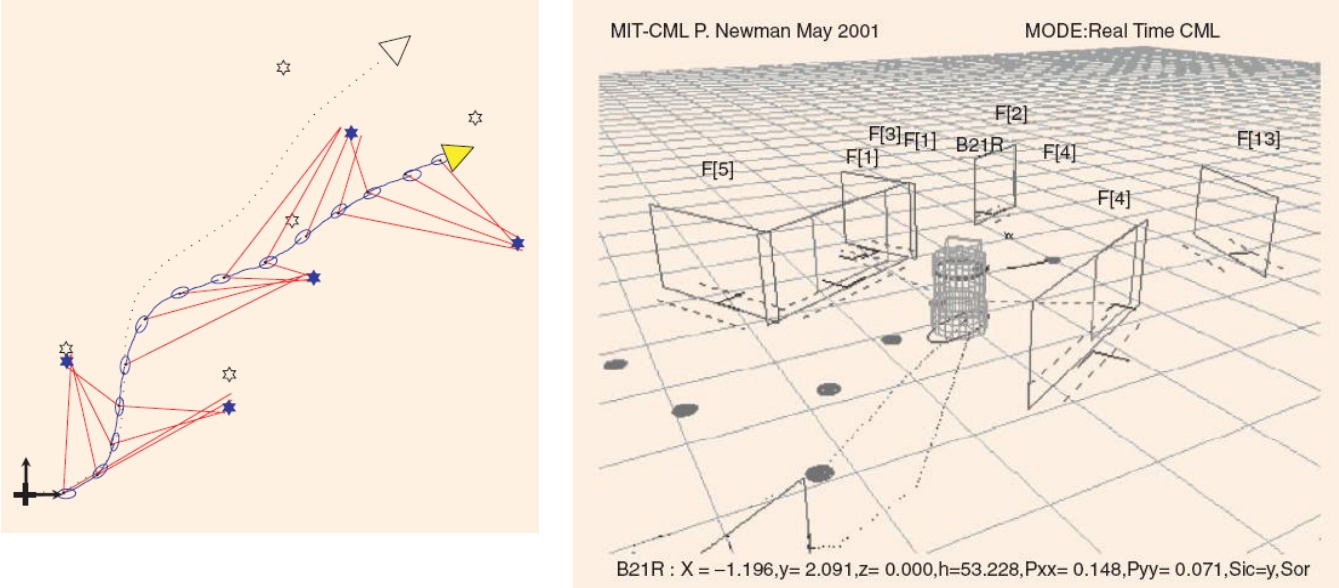
*SLAM* – синхронное определение местоположения и составление карты. Эта задача связана с построением карты неизвестного пространства мобильным роботом во время навигации по строящейся карте. *Исходные данные*: некоторая среда (помещение, открытая местность, водная или воздушная среда) и робот с датчиками. *Требуется*: в каждый момент времени необходимо знать местоположение робота и карту той части окружающего пространства, которая была доступна для наблюдения с начального момента времени до текущего. Местоположение робота логично требовать в системе локальных координат, связанных с исходным положением робота, так как априорной информации об его положении относительно пространства нет. Требования к построенной карте могут быть весьма разнообразны. Главным из них является возможность робота ориентироваться по ней. Без этого вычисление местоположения и построение карты будут осуществляться независимо, что приведет к непрерывному росту погрешности. Другими требованиями могут быть возможность навигации, доступность для человеческого восприятия, поддержка больших пространств и т. д.

Приведем ряд пояснительных примеров к задаче.

На рис. 1, а изображены две траектории – реальная траектория робота (пунктиром) и траектория, вычисленная алгоритмом (синяя линия). Промежуточные положения робота указаны синими эллипсами. Ориентиры изображены звездами: реальные – прозрачными, а вычисленные – синими. По рисунку видно, как накапливается погрешность на каждом следующем шаге. В данном случае построенной картой является *2D*-карта ориентиров.

Рис. 1, б показывает одну из реализаций построения карты препятствий. В качестве датчиков робота использовались *2D*-лазерный дальномер и одометр. Подобная карта может применяться для навигации. При использовании данной карты для ориентации робота требуется достаточно точная информация о его местоположении. При этом появляется возможность нахождения соответствия между види-

мыми препятствиями (их формой, размером, ориентацией) и препятствиями на карте.




а Б

Рис. 1. Одна из траекторий, вычисленных с помощью алгоритма *FastSLAM* [4] (а). Карта препятствий, полученная в режиме реального времени с использованием лазерного дальномера и одометра (б).

В настоящее время методы решения задачи *SLAM* быстро развиваются. Это обусловлено увеличением производительности вычислительных машин, улучшением качества датчиков и появлением новых, развитием робототехники. Задача *SLAM* является очень важной, так как без ее решения вряд ли возможно создание по-настоящему автономного робота.

Задача *SLAM* состоит из множества подзадач. Как правило, выделяются следующие основные подзадачи:

1. Рекурсивный фильтр [5].
2. Нахождение ориентиров в пространстве.
3. Поиск соответствий между ориентирами.
4. Пересчет положения робота.
5. Уточнение положения ориентиров на карте.

### 1.1. Стандартная схема алгоритма

При всем многообразии применяемых в задаче синхронного определения местоположения и построения методов, можно выделить некоторую классическую, весьма общую схему работы алгоритма [2]. Она приведена на рис. 2.

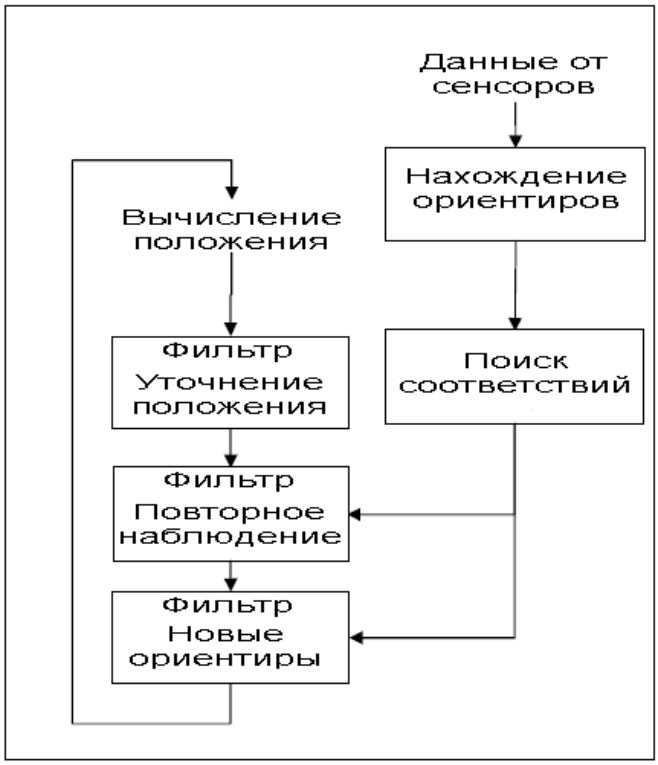



Рис. 2. Классическая схема алгоритма *SLAM*

На каждом шаге алгоритма на вход подаются данные от сенсоров. По этим данным в пространстве находятся ориентиры и определяются их описания, необходимые для поиска соответствий. В процессе работы строится структура, которая хранит ориентиры и их описания. При поиске соответствий для каждого обнаруженного ориентира ищется соответствие в этой структуре. Если соответствие не найдено, то ориентир просто добавляется в структуру. Если же было обнаружено соответствие, то ориентир используется для вычисления положения.

После вычисления положения применяется рекурсивный фильтр для снижения уровня шума в вычисленном положении (уточнение положения). Затем, для обнаруженных ориентиров, у которых нашлись соответствия, выполняется уточнение их положения с помощью рекурсивного фильтра.

## 1.2. Рекурсивные фильтры

В задаче *SLAM* фильтры используются для получения точных, непрерывно обновляемых оценок положения и скорости некоторого объекта по результатам временного ряда неточных измерений его местоположения. Также фильтры применяются для уточнения положения пространственных ориентиров.

### 1.2.1. Фильтр Калмана

Фильтры Калмана [6, 7] базируются на дискретных по времени линейных динамических системах. Так как движение автономных роботов плохо аппроксимируется линейным движением, то фильтры Калмана не получили широкого распространения в реализациях задачи *SLAM*. В то же время, возможно применение фильтров Калмана для вычисления положения неподвижных пространственных ориентиров, так как они удовлетворяют свойству линейности.

При использовании фильтра Калмана для получения оценок вектора состояния процесса по серии зашумленных измерений необходимо представить модель данного процесса в соответствии со структурой фильтра – в виде матричного уравнения определенного типа. Для каждого такта $k$ работы фильтра необходимо в соответствии с приведенным ниже описанием определить матрицы: эволюции процесса $\mathbf{F}_k$; матрицу наблюдений $\mathbf{H}_k$; ковариационную матрицу процесса $\mathbf{Q}_k$; ковариационную матрицу шума измерений $\mathbf{R}_k$; при наличии управляющих воздействий матрицу их коэффициентов $\mathbf{B}_k$.

Модель системы/процесса подразумевает, что истинное состояние в момент $k$ получается из истинного состояния в момент $(k-1)$ в соответствии с уравнением:

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k,$$

где

- $\mathbf{F}_k$ – матрица эволюции процесса/системы, которая воздействует на вектор состояния в момент $(k-1)$ $\mathbf{x}_{k-1}$;

- $\mathbf{B}_k$ – матрица управления, которая прикладывается к вектору управляющих воздействий $\mathbf{u}_k$;
- $\mathbf{w}_k$ – нормальный случайный процесс с нулевым математическим ожиданием и ковариационной матрицей $\mathbf{Q}_k$, который описывает случайный характер эволюции системы/процесса:

$$\mathbf{w}_k \sim N(0, \mathbf{Q}_k).$$

В момент $k$ производится наблюдение (измерение) $\mathbf{z}_k$ истинного вектора состояния $\mathbf{x}_k$, которые связаны между собой уравнением:

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k ,$$

где $\mathbf{H}_k$ – матрица измерений, связывающая истинный вектор состояния и вектор произведенных измерений, $\mathbf{v}_k$ – белый гауссовский шум измерений с нулевым математическим ожиданием и ковариационной матрицей $\mathbf{R}_k$:

$$\mathbf{v}_k \sim N(0, \mathbf{R}_k).$$

Начальное состояние и векторы случайных процессов на каждом такте $\{\mathbf{x}_0, \mathbf{w}_1, ..., \mathbf{w}_k, \mathbf{v}_1 ... \mathbf{v}_k\}$ считаются независимыми.

Многие реальные динамические системы нельзя точно описать данной моделью. На практике не учтенная в модели динамика может серьезно испортить рабочие характеристики фильтра, особенно при работе с неизвестным стохастическим сигналом на входе. Более того, неучтенная в модели динамика может сделать фильтр неустойчивым. С другой стороны, независимый белый шум в качестве сигнала не будет приводить к расхождению алгоритма. Задача отделения шумов измерений от неучтенной в модели динамики сложна, решается она с помощью теории робастных систем управления.

Для вычисления оценки состояния системы на текущий такт работы фильтру Калмана необходимы оценка состояния (в виде оценки состояния системы и оцен-

ки погрешности определения этого состояния) на предыдущем такте работы и измерения на текущем такте.

Далее под записью $\hat{\mathbf{x}}_{n|m}$ будем понимать оценку истинного вектора $\mathbf{x}$в момент *n* с учетом измерений с момента начала работы и по момент *m* включительно.

Состояние фильтра задается двумя переменными:

- $\hat{\mathbf{x}}_{k|k}$ – апостериорная оценка состояния объекта в момент *k*, полученная по результатам наблюдений вплоть до момента *k* включительно (в русскоязычной литературе часто обозначается $\hat{\mathbf{x}}_k$, где $\hat{}$ означает «оценка», а *k* – номер такта, на котором она получена);
- $\mathbf{P}_{k|k}$ – апостериорная ковариационная матрица ошибок, задающая оценку точности полученной оценки вектора состояния и включающая в себя оценку дисперсий погрешности вычисленного состояния и ковариации, показывающие выявленные взаимосвязи между параметрами состояния системы (в русскоязычной литературе часто обозначается $\hat{\mathbf{D}}_k$).

Итерации фильтра Калмана делятся на две фазы: экстраполяция и коррекция. Во время экстраполяции фильтр получает предварительную оценку состояния системы $\hat{\mathbf{x}}_{k|k-1}$ (в русскоязычной литературе часто обозначается $\tilde{\mathbf{x}}_k$, где $\tilde{}$ означает «экстраполяция», а *k* – номер такта, на котором она получена) на текущий шаг по итоговой оценке состояния с предыдущего шага (либо предварительную оценку на следующий такт по итоговой оценке текущего шага, в зависимости от интерпретации). Эту предварительную оценку также называют априорной оценкой состояния, так как для ее получения не используются наблюдения соответствующего шага. В фазе коррекции априорная экстраполяция дополняется соответствующими текущими измерениями для коррекции оценки. Скорректированная оценка также называется апостериорной оценкой состояния, либо просто оценкой вектора состояния $\hat{\mathbf{x}}_k$. Обычно эти две фазы чередуются: экстраполяция производится по результатам коррекции до следующего наблюдения, а коррекция производится совместно с дос-

тупными на следующем шаге наблюдениями, и т. д. Однако возможно и другое развитие событий, если по некоторой причине наблюдение оказалось недоступным, то этап коррекции может быть пропущен и выполнена экстраполяция по нескорректированной оценке (априорной экстраполяции). Аналогично, если независимые измерения доступны только в отдельные такты работы, то все равно возможны коррекции (обычно с использованием другой матрицы наблюдений $\mathbf{H}_k$).

Далее рассмотрим работу классического оптимального фильтра Калмана.

**Этап экстраполяции**

Экстраполяция (предсказание) вектора состояние системы по оценке вектора состояния и примененному вектору управления с шага ($k-1$) на шаг $k$:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k\mathbf{u}_{k-1}.$$

Ковариационная матрица для экстраполированного вектора состояния:

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k\mathbf{P}_{k-1|k-1}\mathbf{F}_k^T + \mathbf{Q}_{k-1}.$$

**Этап коррекции**

Отклонение полученного на шаге $k$ наблюдения от наблюдения, ожидаемого при произведенной экстраполяции:

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1}.$$

Ковариационная матрица для вектора отклонения (вектора ошибки):

$$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k.$$

Оптимальная по Калману матрица коэффициентов усиления, формирующаяся на основании ковариационных матриц, имеющейся экстраполяции вектора состояния и полученных измерений (посредством ковариационной матрицы вектора отклонения):

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{S}_k^{-1}.$$

Коррекция ранее полученной экстраполяции вектора состояния – получение оценки вектора состояния системы:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k.$$

Расчет ковариационной матрицы оценки вектора состояния системы:

$$\mathbf{P}_{k|k} = (I - \mathbf{K}_k \mathbf{H}_k)\mathbf{P}_{k|k-1}.$$

Выражение для ковариационной матрицы оценки вектора состояния системы справедливо только при использовании приведенного оптимального вектора коэффициентов. В общем случае это выражение имеет более сложный вид.

Несмотря на то, что фильтр Калмана применим только к линейным системам, и не подходит для коррекции положения автономного робота, он вполне подходит и эффективен для оценок положения неподвижных пространственных точек.

### 1.2.2. Расширенный фильтр Калмана

Расширенный фильтр Калмана [7] (*Extended Kalman Filter – EKF*) является нелинейной версией фильтра Калмана. Одно время *EKF* мог считаться стандартом в теории нелинейного вычисления положения, навигационных систем и *GPS*. Однако с появлением ансцентного фильтра Калмана (*unscented Kalman filter – UKF*), положение изменилось.

В расширенном фильтре Калмана состояние системы и наблюдения не обязаны быть линейными функциями состояния, а должны быть дифференцируемыми:

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) + \mathbf{w}_{k-1};$$

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k.$$

Основная идея, применяемая в расширенном фильтре Калмана, состоит в приближении функций состояния и наблюдения с использованием их первых производных.

**Этап экстраполяции**

Экстраполяция (предсказание) вектора состояние системы по оценке вектора состояния и примененному вектору управления с шага ($k-1$) на шаг $k$:

$$\hat{\mathbf{x}}_{k|k-1} = f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_{k-1}).$$

Ковариационная матрица для экстраполированного вектора состояния:

$$\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^{\top} + \mathbf{Q}_{k-1}.$$

**Этап коррекции**

Красным цветом выделены отличия от фильтра Калмана.

Отклонение полученного на шаге $k$ наблюдения от наблюдения, ожидаемого при произведенной экстраполяции:

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1}).$$

Ковариационная матрица для вектора отклонения (вектора ошибки):

$$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^{\top} + \mathbf{R}_k.$$

Оптимальная по Калману матрица коэффициентов усиления, формирующаяся на основании ковариационных матриц, имеющейся экстраполяции вектора состояния и полученных измерений (посредством ковариационной матрицы вектора отклонения):

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^{\top}\mathbf{S}_k^{-1}.$$

Коррекция ранее полученной экстраполяции вектора состояния – получение оценки вектора состояния системы:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k.$$

Расчет ковариационной матрицы оценки вектора состояния системы:

$$\mathbf{P}_{k|k} = (I - \mathbf{K}_k \mathbf{H}_k)\mathbf{P}_{k|k-1}.$$

Матрицы изменения состояния системы и наблюдения определяются Якобианами:

$$\mathbf{F}_{k-1} = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_{k-1}}$$

$$\mathbf{H}_k = \left.\frac{\partial h}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k|k-1}}.$$

**Недостатки**

Расширенный фильтр Калмана в общем случае не является оптимальным, в отличие от линейного аналога. Также, если начальное вычисление состояния системы ошибочно, или процесс смоделирован некорректно, результаты могут быстро расходиться из-за линеаризации. Еще одной проблемой является то, что вычисляемая матрица ковариации склонна к недооценке реальной ковариации и, поэтому, риски становятся мало связанными с реальной ситуацией без добавления «стабилизирующего шума».

### 1.2.3. Ансцентный фильтр Калмана

Если функция пересчета состояния системы и функция наблюдений сильно нелинейные, расширенный фильтр Калмана дает плохие результаты. Это связано с вычислением ковариации с помощью линеаризации нелинейной модели. Ансцентный фильтр Калмана [8 – 10] (*UK*-фильтр) использует детерминированную выборку, известную как ансцентное преобразование (the unscented transform) для выбора минимального набора точек (называемых сигма-точками) вокруг среднего значения. Сигма-точки пропускаются через нелинейные функции, по которым затем восстанавливается среднее значение и ковариация.

В результате получается фильтр, который более точно фиксирует среднее значение и ковариацию. Помимо этого, подобный подход снимает требование на вычисление Якобианов, что для сложных функций может представлять серьезную проблему.

**Этап экстраполяции**

Вычисляемое состояние и ковариация расширяются с помощью среднего значения и ковариации шума рассматриваемого процесса:

$$\mathbf{x}^a_{k-1|k-1} = [\hat{\mathbf{x}}^T_{k-1|k-1} \quad E[\mathbf{w}^T_k]\ ]^T$$

$$\mathbf{P}^a_{k-1|k-1} = \begin{bmatrix} \mathbf{P}_{k-1|k-1} & 0 \\ 0 & \mathbf{Q}_k \end{bmatrix}.$$

Набор из *2L + 1* сигма-точек находится из расширенного состояния и ковариации, *L* – размерность расширенного состояния системы:

$$\chi^0_{k-1|k-1} = \mathbf{x}^a_{k-1|k-1};$$

$$\chi^i_{k-1|k-1} = \mathbf{x}^a_{k-1|k-1} + \left(\sqrt{(L+\lambda)\mathbf{P}^a_{k-1|k-1}}\right)_i \quad i = 1..L;$$

$$\chi^i_{k-1|k-1} = \mathbf{x}^a_{k-1|k-1} - \left(\sqrt{(L+\lambda)\mathbf{P}^a_{k-1|k-1}}\right)_{i-L} i = L+1,\ldots 2L,$$

где

$\left(\sqrt{(L+\lambda)\mathbf{P}^a_{k-1|k-1}}\right)_i$ – столбец номер *i* матрицы квадратного корня из $(L+\lambda)\mathbf{P}^a_{k-1|k-1}$.

По определению квадратный корень *A* матрицы *B* удовлетворяет тождеству:

$$B \equiv AA^T$$

Матричный квадратный корень должен вычисляться эффективными и устойчивыми методами, такими как разложение Холецкого [11].

Сигма-точки преобразуются функцией переноса (transition function) $f$:

$$\chi^i_{k|k-1} = f(\chi^i_{k-1|k-1}) \quad i = 0..2L.$$

Из взвешенных сигма-точек строится линейная комбинация для получения экстраполяции состояния системы и ковариации:

$$\hat{\mathbf{x}}_{k|k-1} = \sum_{i=0}^{2L} W_s^i \chi_{k|k-1}^i$$

$$\mathbf{P}_{k|k-1} = \sum_{i=0}^{2L} W_c^i \ [\chi_{k|k-1}^i - \hat{\mathbf{x}}_{k|k-1}][\chi_{k|k-1}^i - \hat{\mathbf{x}}_{k|k-1}]^T.$$

Веса для состояний и ковариации получаются следующим образом:

$$W_s^0 = \frac{\lambda}{L+\lambda}$$

$$W_c^0 = \frac{\lambda}{L+\lambda} + (1 - \alpha^2 + \beta)$$

$$W_s^i = W_c^i = \frac{1}{2(L+\lambda)}$$

$$\lambda = \alpha^2(L+\kappa) - L.$$

Параметры $\alpha$ и $\kappa$ контролируют распределение сигма точек. Параметр $\beta$ отвечает за распределение $x$. Стандартные значения: $\alpha = 10^{-3}$, $\kappa = 0$ и $\beta = 2$. Если реальное распределение $x$ является Гауссовым, то $\beta = 2$ оптимально [12].

**Этап коррекции**

Предсказанное состояние и ковариация расширяются, как и прежде, только используется среднее и ковариация измеренного шума:

$$\mathbf{x}_{k|k-1}^a = [\hat{\mathbf{x}}_{k|k-1}^T \quad E[\mathbf{v}_k^T] \ ]^T;$$

$$\mathbf{P}_{k|k-1}^a = \begin{bmatrix} \mathbf{P}_{k|k-1} & 0 \\ 0 & \mathbf{R}_k \end{bmatrix}.$$

Как и прежде набор из $2L + 1$ сигма-точек вычисляется по расширенному состоянию и ковариации, $L$ – размерность расширенного состояния:

$$\chi_{k|k-1}^0 = \mathbf{x}_{k|k-1}^a$$

$$\chi_{k|k-1}^i = \mathbf{x}_{k|k-1}^a + \left(\sqrt{(L+\lambda)\mathbf{P}_{k|k-1}^a}\right)_i \qquad i = 1..L$$

$$\chi_{k|k-1}^i = \mathbf{x}_{k|k-1}^a - \left(\sqrt{(L+\lambda)\mathbf{P}_{k|k-1}^a}\right)_{i-L} \quad i = L+1, \ldots 2L.$$

Если же этап экстраполяции *UKF* использовал сигма-точки без расширения, то возможно следующее расширение:

$$\chi_{k|k-1} := [\chi^T_{k|k-1} \quad E[\mathbf{v}^T_k]\ ]^T \pm \sqrt{(L+\lambda)\mathbf{R}^a_k}$$

$$\mathbf{R}^a_k = \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{R}_k \end{bmatrix}.$$

Сигма-точки проектируются с помощью функции наблюдения *h*:

$$\gamma^i_k = h(\chi^i_{k|k-1}) \quad i = 0..2L$$

Из взвешенных сигма точек создается линейная комбинация для получения предсказанного измерения и предсказанной ковариации:

$$\hat{\mathbf{z}}_k = \sum_{i=0}^{2L} W^i_s \gamma^i_k$$

$$\mathbf{P}_{z_k z_k} = \sum_{i=0}^{2L} W^i_c\ [\gamma^i_k - \hat{\mathbf{z}}_k][\gamma^i_k - \hat{\mathbf{z}}_k]^T.$$

Матрица взаимной ковариации измеренного состояния

$$\mathbf{P}_{x_k z_k} = \sum_{i=0}^{2L} W^i_c\ [\chi^i_{k|k-1} - \hat{\mathbf{x}}_{k|k-1}][\gamma^i_k - \hat{\mathbf{z}}_k]^T$$

используется для вычисления Калманового прироста *UK*-фильтра:

$$K_k = \mathbf{P}_{x_k z_k} \mathbf{P}^{-1}_{z_k z_k}.$$

Так же как и в случае фильтра Калмана, обновленное состояние – это предсказанное состояние плюс взвешенная добавка:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + K_k(\mathbf{z}_k - \hat{\mathbf{z}}_k).$$

Обновленная ковариация – это предсказанная ковариация минус матрица предсказанной ковариации измерения, взвешенной приращением Калмана:

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - K_k \mathbf{P}_{z_k z_k} K^T_k.$$

### 1.3. Датчики

Реализация *SLAM* невозможна без датчиков, которые дают информацию об окружающем пространстве. Основное внимание будет уделено рассмотрению дат-

чиков применительно к колесным роботам. Роботы с более сложной структурой движения (человекоподобные роботы, автономные подводные аппараты, беспилотные летательные аппараты) в данной работе не рассматриваются.

### 1.3.1. Дальномеры

При применении датчиков, измеряющих дальность, в первую очередь, речь идет о лазерных *2D*-сканерах. Они позволяют получать расстояние до препятствий одновременно для большого угла обзора (до 120 градусов и более). Сканеры очень точны, эффективны и их выходные данные не требуют сложной обработки. В то же время они весьма дорогие, их стоимость может составлять порядка 5000 USD и более. Кроме этого возникает ряд проблем с их использованием в местах, где есть стекло (из-за отражений) и под водой (из-за рассеивания).

Вторым вариантом является применение сонаров. Несколько лет назад они применялись весьма широко. Их стоимость значительно ниже, чем у лазерных сканеров. Однако их изменения не настолько точны и часто происходят неправильные считывания из-за внешних помех. В то время как у сканеров ширина сканирующей линии составляет порядка 0.25 градуса, у сонаров сигнал в ширину может достигать 30 градусов. Под водой сонар является предпочтительным вариантом использования, являясь аналогией навигации дельфинов. Сонары были успешно применены в работе [13].

### 1.3.2. Одометры

Одометры позволяют определять перемещения и повороты робота по положению и вращению колес. Для их использования желательной является точность не хуже двух сантиметров на метр прямолинейного перемещения и не хуже двух градусов на 45 градусов угла поворота колес.

Одной из проблем одометров является скольжение и пробуксовка колес робота. Они возникают при резкой остановке, при движении по скользким или неоднородным покрытиям, при встрече препятствия. При отсутствии других датчиков

решить данную проблему не представляется возможным. В то же время использование одометра широко распространено в качестве дополнения к другим датчикам (например, комбинация лазерный дальномер плюс одометр).

### 1.3.3. Видео камеры

Традиционно применение видео камер является вычислительно сложным и склонным к ошибкам из-за изменения уровня освещенности. В отсутствии света использование камер также затруднительно. В последнее время в данной области произошел ряд достижений. Стерео системы применяются для измерения расстояний. Помимо этого, при использовании видео появляется много дополнительной информации. Возможными проблемами при применении камер являются: однородная окружающая среда, отражающие поверхности, неточности калибровки. Измерение расстояний по видео было успешно реализовано в работе [14].

## 1.4. Детекторы точек

При применении видео камер требуется решать задачу поиска соответствующих точек на изображениях с левой и с правой камеры. Она состоит из двух частей: поиск точек интереса (interest points, key point, ключевая точка) и нахождение соответствий между ними. Как правило, используются точки, находящиеся в экстремумах яркости изображения. Для каждой найденной точки строится дескриптор и затем осуществляется поиск соответствий, где критерием совпадения является малое расстояние дескрипторов точек.

Помимо поиска соответствий на двух изображениях, к найденным точкам выдвигаются дополнительные требования: устойчивость к растяжению, повороту изображений, к шуму, к изменению точки наблюдения. Также дескрипторы различных точек должны отличаться достаточно сильно для получения возможности осуществлять поиск точек по большой базе.

Существуют разные методы, но применительно к задаче *SLAM* наибольшее распространение получили методы *SIFT* (*Scale-Invariant Feature Transform*) [15 – 17] и *SUFR* (*Speeded-Up Robust Features*) [18, 19].

### *1.4.1. SIFT*

Точки интереса, которые находятся с помощью детектора *SIFT*, устойчивы к растяжению, повороту изображения и частично к изменению точки наблюдения. Для стандартных изображений получается большое число точек, описания (дескрипторы) которых значительно отличаются, что делает возможным поиск по большой базе точек.

Стоимость нахождения точек снижается применением пошаговой фильтрации по нескольким признакам, в которой более затратные критерии проверяются только в случае прохождения первоначального теста.

Основные этапы генерации набора точек изображения:

1. Нахождение экстремумов по всем шкалам и точкам изображений. Реализуется путем вычисления разности Гауссовых функций [20], что позволяет находить потенциально интересные точки, которые инвариантны по отношению к растяжению и поворотам.
2. Локализация ключевых точек. В каждой точке, найденной в п.1, строится детализированная модель для уточнения положения ключевой точки и ее размера.
3. Добавление ориентации. Одна или несколько ориентаций добавляются к каждой точке на основании направлений градиентов. В дальнейшем все операции производятся над полученным положением, размеров и ориентациями точки. Это позволяет получить устойчивость к поворотам и растяжениям изображения.
4. Дескрипторы ключевых точек. Локальные градиенты изображения измеряются по выбранной шкале в районе каждой ключевой точки. Эти данные

преобразуются к виду, допускающему значительную степень изменения формы и изменения освещения.

При размере изображения порядка 500*500, *SIFT* позволяет получать порядка 2000 устойчивых точек (хотя следует отметить, что это число сильно зависит от содержимого изображения и параметров алгоритма). Для каждой ключевой точки строится дескриптор размером в 128 вещественных чисел.

### *1.4.2. SURF*

Детектор точек *SURF* [21] был предложен Гербертом Бэйем (Herbert Bay) в 2006 году. В некоторой степени он использует идеи, примененные в детекторе точек *SIFT*. В то же время, стандартная версия *SURF* в несколько раз быстрее, чем *SIFT*. При этом детектор строит несколько меньшее число точек интереса, но они более устойчивы по отношению к различным преобразованиям изображения.

В *SURF* применяются суммы приближенных *2D Haar wavelet* значений и эффективное использование интегральных изображений. Подробное изложение метода можно найти в работе [18].

### 1.4.3. Поиск соответствий между точками

Для поиска соответствий между точками интереса, как правило, применяется *kd*-дерево [22]. У каждой ключевой точки имеется дескриптор, состоящий из *d* вещественных чисел (у детектора *SIFT* $d = 128$, у детектора *SURF* $d = 64$). При поиске соответствий между двумя наборами точек по одному из них строится *kd*-дерево и для каждой точки из второго набора осуществляется поиск соответствия в построенном *kd*-дереве. При поиске соответствия для точки C в *kd*-дереве осуществляется поиск двух ближайших точек (*A* и *B*) по расстоянию между дескрипторами. Если расстояние от точки C до точки *B* превышает расстояние от точки C до точки *A* в два и более раз, и расстояние между *C* и *A* достаточно мало, то считается, что найдено соответствие между точкой *C* и *A*. В противном случае, соответствие считается необнаруженным.

Эффективность поиска по *kd*-дереву снижается при большой размерности пространства поиска. Поэтому в подобных пространствах применяется приближенный алгоритм поиска по *kd*-дереву *Best Bin First* (*BBF*) [23]. Идея этого алгоритма состоит в выборе очередности поиска – вначале в поддеревьях с меньшим отклонением от точки поиска, а также в прекращении поиска после просмотра некоторого фиксированного числа листьев *kd*-дерева (обычно порядка 200). В большом числе случаев *BBF* находит ближайшую к искомой точке точку, а в остальных случаях находит достаточно близкую.

*Kd*-дерево строится рекурсивно. При построении *kd*-дерева определяется координата *m*, по которой у *k*-мерных точек наилучшее распределение. Затем, по всем *m*-координатам точек ищется медиана и производится разбиение точек на два множества – на точки, у которых *m*-координата меньше медианы и у которых больше. Затем для получившихся подмножеств строятся *kd*-деревья, и они становятся левым и правым поддеревьями искомого дерева. Таким образом, получается сбалансированное дерево. Указанные особенности построения *kd*-дерева не позволяют его модифицировать напрямую (хотя если считать, что на вход подаются случайные данные, то подход, приведенный в работе [22], поддерживает как минимум добавление в *kd*-дерево).

## Выводы по главе 1

1. Обзор поставленной задачи и применяемых для ее решения методов показывает, что сама по себе задача *SLAM* весьма сложна и ее подзадачи охватывают множество областей (компьютерное зрение, теория вероятности, вычислительная геометрия и т.д.).
2. В главе были рассмотрены подзадачи и методы их решения применительно к реализации *SLAM* на основе стерео изображений.
3. Приведено сравнение некоторых методов реализации подзадач с целью обоснования их применения в реализованном решении.

# 2. ИЗВЕСТНЫЕ РЕАЛИЗАЦИИ

Большая часть успешных реализаций *SLAM* используют высокоточную технику. Как правило, применяются роботы с высокоточным одометром, *2D*-лазерным сканером или стерео парой с высокоточной калибровкой. Подобная аппаратура позволяет получать точные локальные перемещения, обнаруживать препятствия. В то же время, стоимость роботов, оснащенных такими датчиками, высока.

Так как в работе в качестве датчиков применяются видео камеры, то будем рассматривать, в первую очередь, реализации, использующие видео.

## 2.1. Система трех камер в виртуальном пространстве

В работе [24] реализуется алгоритм *FastSLAM* [4]. Хотя, первоначально *FastSLAM* был разработан для использования лазерных сканеров и построения *2D*-карты, было показано, что он может быть адаптирован для трехмерного случая и использования камер. В работе применяется виртуальная среда и виртуальная тринокулярная система камер. Постановка задачи подразумевает шесть степеней свободы у системы камер. Карта строится на основе наблюдаемых пространственных линий. Предложенное решение, помимо хранения набора гипотез о положении камер на каждой итерации, хранит и наборы карт, соответствующих указанным гипотезам. Данный подход имеет хорошие шансы на преодоление систематических ошибок в построении карты, вносимых неточным вычислением соответствий.

Ниже на рис. 3 и 4 приведены примеры изображений, генерируемых для алгоритма и результаты построения карты в различных условиях.

а б В

Рис. 3. Изображения с тринокулярной системы камер «центральная» камера (а), камера со сдвигом влево (б) и камера со сдвигом вверх (в)

В процессе тестирования эффективности алгоритма, в данной работе рассматривались три случая (рис. 3, 4 все данные в метрах). В первом из них положение камер считалось известным точно (фактически решалась задача построения карты при известном местоположении). Таким образом, проверялась корректность определения местоположения *3D*-линий. Получившаяся карта является весьма точной.

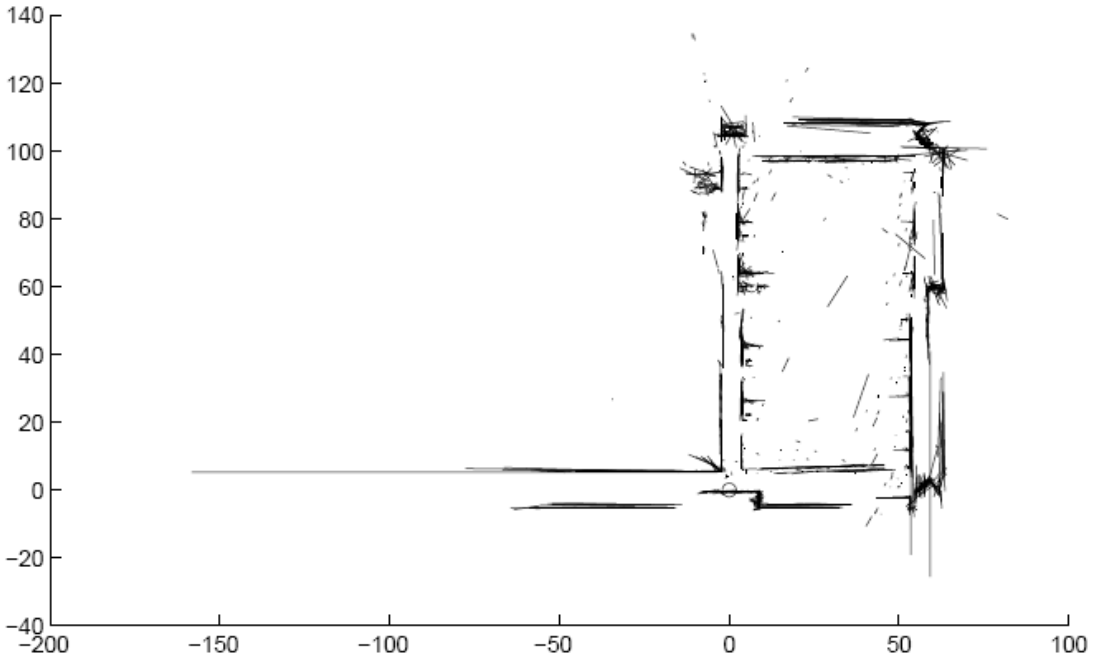
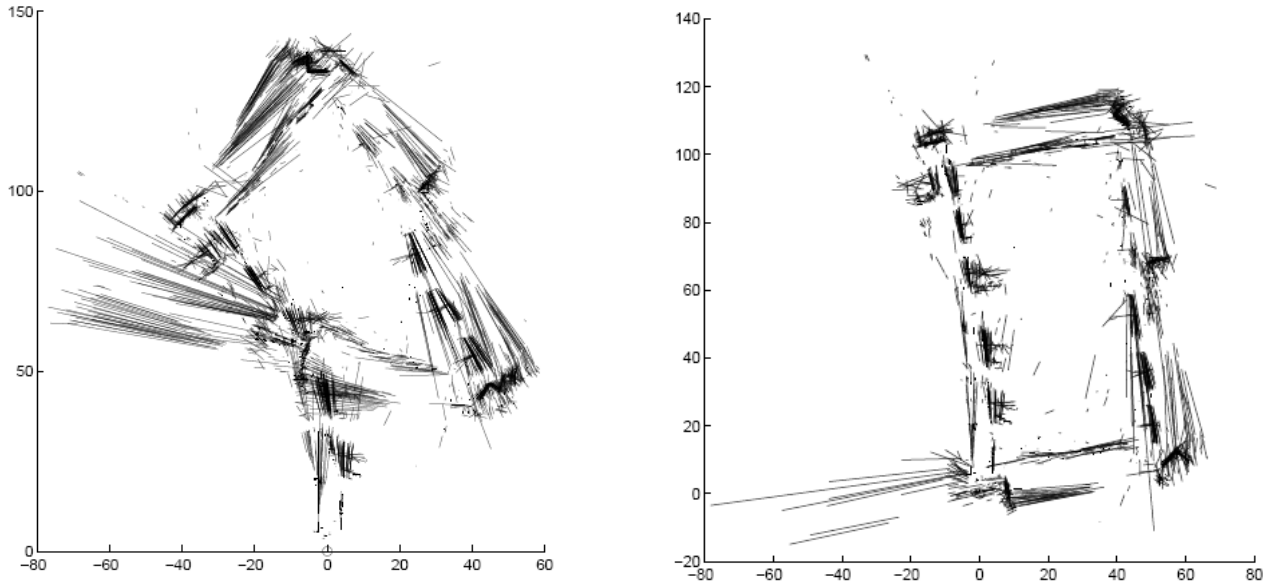



Рис. 4. Карта, построенная с использованием точного местоположения

Во втором случае рассматривался алгоритм, идентичный первому, но для вычисления местоположения использовались только данные одометра (в которых накапливается погрешность). Была получена карта, которая вряд ли может применяться для навигации.

Третий случай – это алгоритм *SLAM*, который и является результатом данной работы. Не обладая большей информацией, чем во втором случае, он строит достаточно качественную карту. Данных о быстродействии в работе [24] не приводится.




а Б

Рис. 5. Карты, построенные без знания точного местоположения. Местоположение определяется по одометру (а), *SLAM* (б).

Отметим и определенные сложности в оценке качества данной работы. Так как изображения и окружающая среда были виртуальными, смоделированными, то велика вероятность отсутствия факторов, негативно влияющих на работу алгоритма: неточность калибровки, изменение освещенности, погрешности оптики камер и другие случайные внешние воздействия.

### *2.2. Гамма-SLAM*

В работе [25] реализуется алгоритм *SLAM* для неструктурированных открытых пространств. В отличие от клеточных алгоритмов, в *Гамма-SLAM* используется

новая техника построения карт, хранящая апостериорное распределение высоты в каждой клетке карты. Для определения совместного апостериорного распределения позиций и карт применяется *Rao-Blackwellized particle filter* [26, 27]. С его помощью вычисляется распределение положения робота, при этом каждой позиции соответствует своя карта.

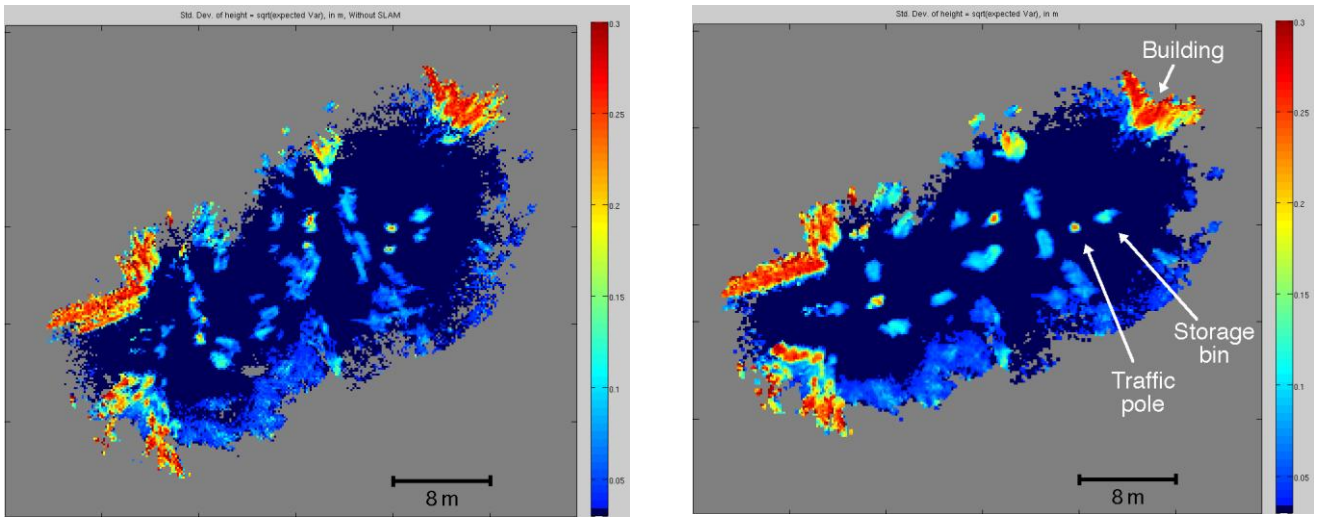
Алгоритм тестировался на модели робота *LAGR hHerminator* [28]. Он обладает акселерометром, двумя стереопарами, которые предоставляют угол обзора в 140 градусов. По изображениям с камер строится карта глубины, а затем она преобразуется в декартовую карту высот. Пересчет положения осуществляется с помощью детектора углов, применяемого к каждому изображению, поиск соответствий использует стерео сумму абсолютных разностей **(***stereo SAD* (*sum of absolute differences*)**)**.

Рис. 6. Фотография места тестирования робота

Карты, приведенные на рис. 7, соответствуют фотографии на рис. 6. Высота отображается цветом – чем выше, тем более яркий цвет.

Результаты данной работы выглядят очень хорошими. Следует также отметить, что робот, использовавшийся в работе, позволяет получать качественные стерео снимки и, как следствие, точные локальные положения пространственных точек.

Также в работе [25] была произведена оценка точности работы алгоритма с помощью *GPS*, но погрешность отдельного измерения *GPS* не позволила произвести достаточно точного сравнения.




а Б

Рис. 7. Карта высот, построенная с применением только визуального ориентирования (а), карта, построенная с использованием 100 гипотез (б)

### 2.3. Другие реализации

Стоит отметить также ряд реализаций, использующих лазерные дальномеры. В работах [27, 29] строятся весьма точные *2D*-карты (рис. 8, 9) больших закрытых пространств. Как было отмечено, датчики, применяемые в этих работах, весьма дорогие. Точность карт высока, время работы алгоритмов неизвестно.

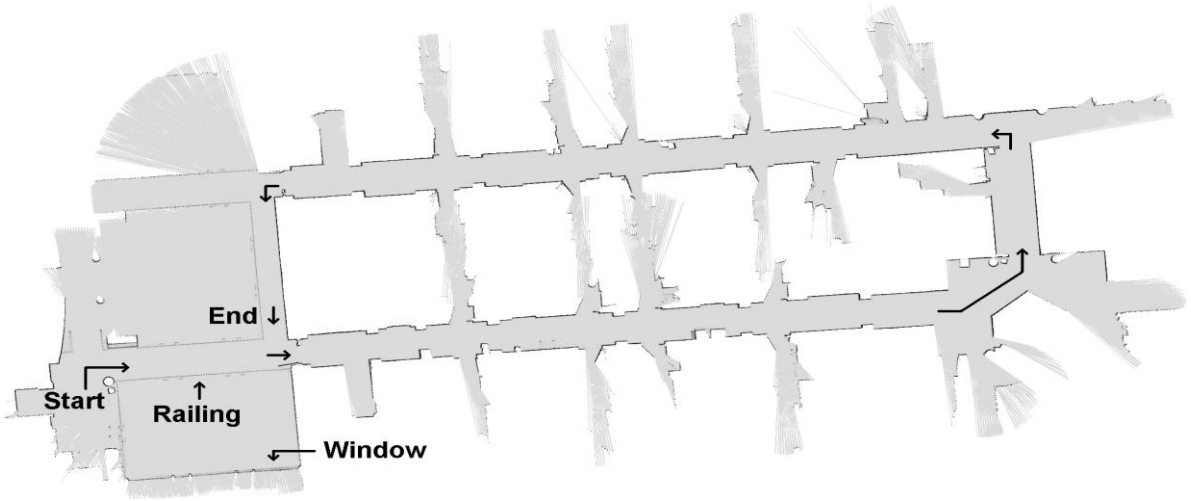



Рис. 8. Карта, полученная алгоритмом *DP-SLAM 2.0* [29]

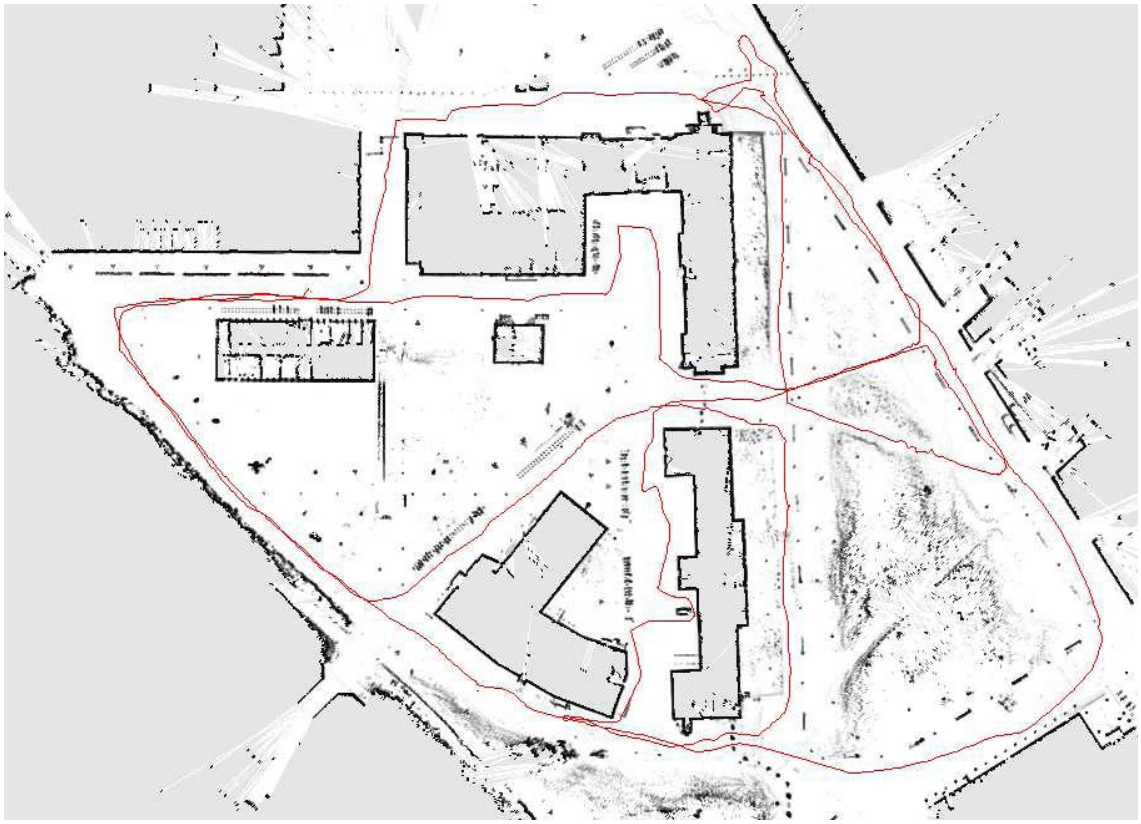


Рис. 9. Карта, полученная с помощью улучшенной техники клеточных карт и *Rao-Blackwellized particle filter*. Размер пространства 250м * 250м

## Выводы по главе 2

1. Обзор работ показывает, что существует ряд весьма точных реализаций *SLAM,* работающих в закрытых пространствах и использующих лазерные *2D*-сканеры.

2. Успешных реализаций, использующих видео для определения местоположения и построения карты, меньше, и они, как правило, менее точные и быстрые. Тем не менее, направление быстро развивается, так как растут как вычислительные мощности, так и доступность качественных камер.

# 3. РЕАЛИЗОВАННЫЙ МЕТОД

В данной работе было реализовано определение местоположения робота по видео, получаемого со стереопары камер, и составление карты пространственных ориентиров для робота.

## 3.1. Постановка задачи

*Задача:* требуется реализовать определение местоположения робота *Ивана* в закрытых помещениях размером до 10*10 м с помощью имеющихся у него датчиков. При этом выдвигаются *требования*: точность определения местоположения не хуже 0.5 м, точность определения угла поворота не хуже 10% от величины угла поворота относительно исходного положения. Также требуется минимизация накопления погрешности – при посещении уже пройденных мест погрешность не должна превышать погрешность, которая была в этих местах ранее. Последнее требование фактически означает то, что при нахождении в уже посещенном месте с совпадающей ориентацией вычисленное местоположение должно совпадать с вычисленным ранее.

## 3.2. Исходные данные

В работе использовался колесный робот *Иван* [32] рис. 10. Он оснащен стерео парой камер разрешением 768*576 пикселей, углом обзора 40 градусов. Одометр отсутствует (расчет перемещения по управляющим командам результата не дал). Таким образом, при решении задачи использовался только один вид датчиков – стерео пара камер. Одной из трудностей, возникших при реализации, стала проблема калибровки камер. В процессе снятия видео калибровка имеет тенденцию изменяться, что приводит к потере точности. В рамках одного измерения потеря точности не слишком существенна, но при серии измерений она накапливается и приводит к существенным ошибкам.

Рис. 10. Колесный робот *Иван*

### 3.3. Нахождение точек-ориентиров

Для нахождения точек-ориентиров применяется детектор точек *SURF* [18, 19] и геометрические соображения. Параметры детектора: изображения не уменьшаются, начальный шаг масштабирования равен двум, число анализируемых октав равно четырем, порог обнаружения экстремумов для построения дескриптора точки равен 0.05. На каждом изображении с камеры размера 576*768 пикселей, при достаточной структурированности среды находится около 2000 точек интереса. Соответствия на другом снимке находятся примерно для 500 точек (в плохих ситуациях может быть менее 100, а иногда и порядка 30).

Для найденных точек интереса требуется вычисление локальных координат. Оно осуществляется из соображений геометрии. Заранее вычисляется матрица переноса от одной камеры к другой с помощью программы калибровки камер. Фокусное расстояние камер и их разрешение также известны. Таким образом, для некоторой точки на изображении камеры появляется возможность построить вектор, соединяющий оптический центр камеры и положение точки на действительном

изображении. Получаем по вектору для каждой камеры для одной точки. Зная эти векторы и преобразование между системами координат камер, вычисляется точка их пересечения (строится вектор, ортогональных обоим векторам и в качестве их пересечения и определяется его середина). Эта точка и является пространственной точкой-ориентиром с уже известными локальными 3D-координатами.

### 3.4. Хранение точек-ориентиров

В данной работе применялось два способа хранения пространственных ориентиров. В связи с тем, что соответствия точек определяются по дескриптору длины *k* (в контексте хранения пространственных ориентиров используем обозначение *k* вместо *d*), а остальная информация не используется, то, абстрагируясь от задачи, будем считать, что требуется хранить только *k*-мерные точки (*k*-точки) – дескрипторы точек-ориентиров. Оба предложенных способа хранения *k*-точек используют не модифицируемые сбалансированные *kd*-деревья.

Первый способ использует одно *kd*-дерево для хранения всех точек. При этом на каждом шаге алгоритма определяется набор точек, которые требуется оставить в дереве и которые требуется добавить в него. Затем эти точки объединяются в один массив, по которому строится новое *kd*-дерево. Старое *kd*-дерево удаляется. Таким образом, происходит модификация структуры для хранения пространственных ориентиров. Преимущество данного подхода состоит в быстром поиске по дереву. Недостатком является плохая асимптотика времени работы – порядка $O(n * \log n + s * \log n)$, где $n$ – число *k*-точек в дереве, а $s$ – число запросов. Проблема состоит в члене $O(n * \log n)$, из-за которого с увеличением числа хранимых точек время работы структуры растет более чем линейно.

Второй способ использует набор из нескольких *kd*-деревьев. Одно такое дерево применяется для хранения не базовых *k*-точек. Это те ориентиры, которые встретились недостаточное число раз (меньше, чем *BaseSeenTimes* раз, использовалось значение равное трем), и первое их появление произошло недавно (прошло не более *MaxNonBaseTime* итераций алгоритма, использовалось значение равное се-

ми). Данное дерево перестраивается на каждой итерации – из него удаляются точки, ставшие базовыми и те, которые не стали базовыми за *MaxNonBaseTime* итераций. Также в него добавляются впервые обнаруженные ориентиры.

Остальные *kd*-деревья образуют отдельную структуру – мульти *kd*-дерево. Оно состоит из нескольких *kd*-деревьев, в которых каждое следующее больше предыдущего. Для каждого дерева задан максимальный размер. При превышении этого размера дерево добавляется к следующему по величине. На каждой итерации поддерживается инвариант – либо *i*-ое дерево пусто, либо размер *i*-ого дерева больше максимального размера предыдущего дерева. При этом размер дерева *i* не превышает максимально допустимого размера *i*-ого дерева. Идея подобной структуры данных состоит в оптимизации добавления множеств *k*-точек. При добавлении набора пространственных ориентиров в мульти *kd*-дерево, они добавляются в первое *kd*-дерево (с наименьшим максимальным размером). Затем происходит восстановление инвариантов. Примерный алгоритм приведен ниже:

```
i = 1;
while (tree[i].size > MAX_SIZE[i])
{
    Объединить деревья с номерами i и i + 1.
    ++i;
}
```

Максимальный размер последнего *kd*-дерева превышает число точек, которые когда-либо могут находиться в мульти *kd*-дереве. Подобный подход позволяет существенно уменьшить время добавления точек в среднем и несколько уменьшить время добавления в худшем по сравнению с одним *kd*-деревом. В то же время, существенно увеличивается время поиска соответствий – приходится искать в нескольких *kd*-деревьях суммарным размером *n*. Пусть размеры деревьев равны $n_1, n_2, n_3 \dots n_p$, $\sum_{i=1}^{p} n_i = n$. Время поиска по *kd*-дереву составляет *O(log n)*, тогда общее время поиска во втором случае составит

$$\sum_{i=1}^{p} O(\log n_i)$$

Это время может быть существенно больше, чем *O(log n),* но не более, чем в *p* раз.

### 3.5. Расчет положения

Рассмотрим вычисление положения на шаге *i*. Вначале находятся пространственные ориентиры с их локальными координатами (система отсчета – текущее положение камер). Затем находится соответствие между ними и ориентирами, находящимися в базе (у них глобальные координаты – относительно исходного положения). После этого применяется алгоритм поиска параллельного переноса и поворота, преобразующего одно трехмерное облако точек в другое [30]. Найденный параллельный перенос и поворот является новым положением и ориентацией камер (и робота, соответственно). Следует отметить, что вычисление сдвига между предыдущим и текущим положением дает худшие результаты, так как задача определения переноса и поворота оказывается плохо обусловленной. Причиной является неточность в определении как локальных, так и глобальных координат пространственных ориентиров, сопоставимая с величиной вычисляемого сдвига и поворота.

Приведенного алгоритма в отдельности не хватает для достаточно точного пересчета координат. При наличии некоторого числа ошибочных соответствий, алгоритм имеет тенденцию к существенным отклонениям в определении положения, достигающим нескольких метров (!). Во избежание подобных ситуаций применяется алгоритм *RANSAC* [31]. Идея этого алгоритма состоит в выборе некоторого случайного множества точек для определения местоположения. Подобный выбор (и вычисление положения) осуществляются несколько раз (например, 200). Затем из полученных положений выбирается наилучшее. Число итераций может быть уменьшено, если удалось быстро найти достаточно хорошее приближение.

Также необходимой является оптимизация, учитывающая, когда были впервые встречены точки, по которым производится пересчет координат. Это очень важно для замыкания цикла и минимизации погрешностей при достижении «знакомых» областей. Логично предположить, что при отсутствии циклов, погрешность имеет тенденцию накапливаться. Таким образом, точность вычисления координат более поздних ориентиров существенно ниже, чем более ранних. Поэтому при определении местоположения имеет смысл ориентироваться на более ранние наблюдения. Пусть имеется набор пространственных ориентиров. Отсортируем его по возрастанию шага, на котором был встречен очередной ориентир. Затем найдем достаточно большое для пересчета местоположения число точек, которые встретились максимально давно и примерно в одно время. По ним и будем осуществлять вычисление позиции робота.

Подобный подход позволяет значительно уменьшить погрешности вычислений в случае движения робота по замкнутой траектории.

### 3.6. Реализация фильтров

Для пространственных ориентиров был реализован фильтр Калмана [6,7], так как стационарные точки вполне отвечают требованиям линейности. Учитывая стационарность точек, было получено, что матрицы $F$ и $H$ должны быть единичными, а матрица $B$ нулевой. Матрицы $R$ и $Q$ были выбраны единичными, а затем умножены на оценку дисперсии состояния системы и погрешности наблюдения соответственно.

Ансцентный фильтр Калмана реализован не был, так как возникли сложности при составлении математической модели движения робота и получении его управляющих параметров. Несмотря на указанные проблемы, реализация такого фильтра является одним из факторов, которые могут улучшить эффективность работы алгоритма.

## 3.7. Оценка точности

Было проведено тестирование программы на наборе тестов: разворот на 360 градусов на месте и движение по замкнутой траектории (примерно прямоугольной формы).

### 3.7.1. Поворот на месте

При повороте на месте разброс вычисленных положений составил порядка 20 см. Часть этого разброса обусловлена неточностью вращения камер вокруг фокуса. Таким образом, следует отметить, что погрешность определения местоположения при вращении на месте не превышает 15 см.

Погрешность измерения угла поворота вычислить сложнее. Проблема заключается в необходимости визуально оценить угол поворота робота по видео и сравнить его с вычисляемыми значениями. Вначале тестирования робот устанавливается перпендикулярно одной из стен помещения. Затем узнать повороты на углы кратные 90 градусам можно будет по расположению других стен.

Была проведена оценка точности при повороте на 180 градусов. При визуальной оценке угла поворота в 180 градусов отклонение вычисленного поворота было в пределах пяти градусов.

При повороте на 360 градусов погрешность минимизируется благодаря обнаружению пространственных точек, которые были видны при угле поворота в ноль градусов. Они используются для корректировки положения и угла поворота. Погрешность обусловлена точностью единичного вычисления позиции и составляет не более двух-трех градусов.

### 3.7.2. Движение по замкнутой траектории

Для измерения точности вычисления ориентации робота была применена оценка точности вычисленного положения границы исследуемой области. В текущей реализации карта препятствий не строится, но на каждой итерации отображаются точки-ориентиры. Граница трассы представляет собой вертикальное огражде-

ние высотой порядка 50 см с наклеенными на него фотографиями поверхности Луны. При нахождении вблизи границы основная часть ориентиров располагается на вертикальном ограждении, поэтому его легко увидеть среди точек-ориентиров, обнаруженных на данной итерации. Так как две противоположные границы трассы параллельны, и в начале тестирования робот был перпендикулярен одной из них, то их проекции на карту должны быть горизонтальны. Как видно из рисунка, противоположная стена не горизонтальна и отклонение составляет порядка 20 градусов. Это следствие ошибки в вычислении ориентации робота. Так как стена должна быть горизонтальна, то реальный угол поворота робота составляет не 180, а 200 градусов (отклонение совпадает с углом наклона границы области). Получается погрешность равная 20 / 200 * 100% = 10%, причем это погрешность недостаточности поворота. Расчетные точки положения и ориентации робота, точки-ориентиры видны на рис. 11.


Рис. 11. Угол отклонения расчетного положения стены от реального

Следует отметить, что при вращении робота в одном направлении данная погрешность суммируется и при траектории типа «раскручивающаяся спираль» в ограниченных условиях видимости может принимать недопустимые размеры. В то же время при поворотах в разных направлениях погрешность ожидается заметно меньшей (тестирование не проводилось ввиду необходимости большого пространства).

На рис. 12 приведены реальные изображения со стереопары, установленной на роботе.

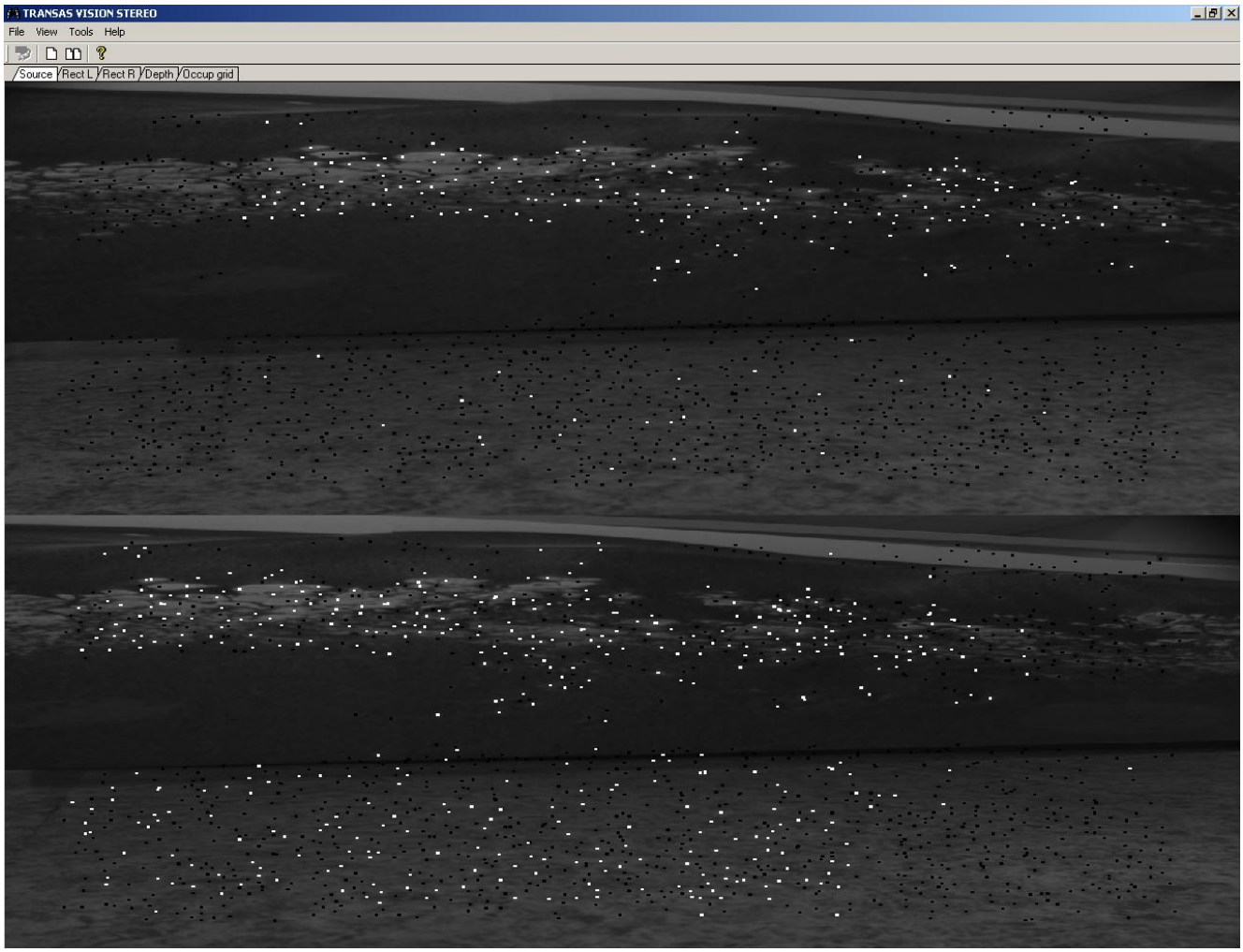



Рис. 12. Изображения с камер – левое вверху, правое внизу

На снимках видна часть пола и граница исследуемой области. Угол отклонения от перпендикуляра к границе порядка 20 градусов. Белыми точками на верхнем изображении отмечены точки, по которым идет пересчет позиции робота. Белым

цветом на нижнем изображении отмечены точки, у которых нашлось соответствие на верхнем снимке (те точки, для которых возможно рассчитать локальные координаты). Черным цветом отмечены точки, обнаруженные детектором точек (все белые точки также обнаружены детектором).

## Выводы по главе 3

1. В данной главе приведен реализованный алгоритм и оценена его точность.
2. Реализация удовлетворяет поставленным требованиям: погрешность определения местоположения не превышает 0.5 метра, при возвращении в посещенные позиции, вычисленное положение совпадает с вычисленным ранее. При повороте в одном направлении погрешность вычисления угла поворота составляет порядка 10%, при повороте в разных направлениях погрешность вычитается (если новая местность). Если робот посещает некоторую местность повторно (с той же ориентацией), то ошибка вычисления местоположения примерно равна ошибке полученной при первом посещении данного места. Это означает, что программа при повторных посещениях выдает те же координаты и ориентацию, что и при первоначальном посещении места.
3. Реализованный алгоритм может работать в режиме реального времени со средней частотой порядка 2,5 Гц. Данной частоты вполне достаточно при небольших угловых скоростях поворота. Для больших скоростей желательно иметь камеры с большим углом обзора, так как при наличии близких препятствий на изображениях соответствующих разным моментам времени мало общих точек.

# 4. СРАВНЕНИЕ ЭФФЕКТИВНОСТИ РЕАЛИЗАЦИЙ ХРАНЕНИЯ *K*-ТОЧЕК

В данной работе рассматривалось две реализации хранения ориентиров: в одном *kd*-дереве и в структуре из нескольких *kd*-деревьев. В обоих случаях применяемая реализация *kd*-деревьев не поддерживала операций добавления или удаления из дерева. Поэтому в первом случае на каждом шаге приходилось строить дерево заново, а во втором – строить заново часть имеющихся деревьев.

## 4.1. Теоретическая оценка времени работы контейнеров для *k*-точек

В данном разделе для некоторого упрощения анализа, абстрагируемся от необходимости удаления *k*-точек из контейнеров. Сделаем теоретическую оценку быстродействия для предложенных вариантов реализации хранения *k*-точек. Размерность пространства точек *k* в оценках учитывать не будем. Будем считать, что на каждой итерации происходит s запросов на поиск *k*-точек и *m* запросов на добавление *k*-точек (для упрощения считаем, что $n \sim n + m$).

### 4.1.1. Не модифицируемое *kd*-дерево

Оценка времени работы не модифицируемого *kd*-дерева не представляет особых сложностей. Как известно, построение *kd*-дерева занимает $O(n * \log n)$, где $n$ – число точек в дереве. На каждой итерации для дерева требуется добавление и удаление элементов, а так как оно не поддерживает эти операции, то время на них равно времени построения дерева. Поиск по дереву занимает $O(\log n)$.

В результате получаем, что время выполнения одной итерации составляет $O(n * \log n + s * \log n)$. Эта оценка справедлива как для среднего, так и для худшего времени.

### 4.1.2. Мульти *kd*-дерево

Оценка времени работы мульти *kd*-дерева заметно сложнее, так как она зависит от ряда параметров. Напомним, что мульти *kd*-дерево поддерживает только добавление *k*-точек. У мульти *kd*-дерева существуют несколько параметров. Первый – максимальный допустимый размер первого *kd*-дерева – *n1*. Второй – во сколько раз возрастает максимальный допустимый размер каждого следующего дерева – *q*. Будем считать, что $n_1 > z * m$, где *z* – натуральное число много большее единицы, например, 10. Данное требование необходимо для того, чтобы избежать ситуации, когда на почти каждой итерации обновляется второе *kd*-дерево, а также для упрощения оценок быстродействия.

Обозначим за *p* наибольший номер непустого дерева. Тогда *n1*, *q* и *p* будут связаны следующим соотношением:

$$\sum_{i=1}^{p} q^{i-1} n_1 \geq n \leftrightarrow \frac{q^p - 1}{q - 1} n_1 \geq n . \qquad (1)$$

В левой части формулы записана сумма максимальных чисел *k*-точек в *kd*-деревьях с первого по *p*-ое. Так как дерево с номером *p* не пусто, то на некоторой итерации произошло переполнение дерева с номером *p-1*. Следовательно,

$$q^{p-1} n_1 \leq n . \qquad (2)$$

Из соотношений (1) и (2) получим:

$$p \geq log_q(\frac{n}{n_1}(q-1)+1)$$

$$p \leq \log_q(\frac{n}{n_1}) + 1 . \qquad (3)$$

Оценим вначале время поиска в мульти *kd*-дереве. Асимптотика совпадает в среднем и в худшем случае. Рассмотрим первые *p* деревьев, кроме них в мульти *kd*-дереве нет непустых *kd*-деревьев. Верхняя оценка времени поиска по *p* деревьям размерами *n1*, *q* * *n1* … составит:

$$\sum_{i=1}^{p} \log\left(q^{i-1}n_1\right) = p * log(n_1) + \sum_{i=1}^{p} (\mathrm{i}-1)\log(q) = p * log(n_1) + \frac{(p-1)p}{2}\log(q)$$
$$= T_{search}(n_1, q);$$

Учитывая (3), можно получить оценку времени поиска сверху, зависящую только от $n$, $n_1$ и $q$:

$$\left(log_q\left(\frac{n}{n_1}\right) + 1\right) * \log(n_1) + \frac{1}{2} log_q\left(\frac{n}{n_1}\right)(log_q\left(\frac{n}{n_1}\right) + 1)\log(q)$$

Остановимся на достигнутом результате и обратимся к оценке среднего времени обновления мульти *kd*-дерева. При любом изменении структуры перестраивается первое дерево, и на это тратится время $O(n_1 \log(n_1))$. На каждой итерации в мульти *kd*-дерево добавляется $m$ точек. Таким образом, в среднем через $\frac{n_1}{m}$ итераций будет происходить переполнение первого *kd*-дерева, и оно будет добавляться ко второму за $O(q * n_1 \log(q * n_1))$. При подобном добавлении размер второго *kd*-дерева будет увеличиваться примерно на $n_1$. Повторяя подобные рассуждения получаются частоты и временные затраты обновлений других *kd*-деревьев. Получаем сумму:

$$\sum_{i=2}^{p} \frac{m}{n_1 * q^{i-2}} * (n_1 q^{i-1} \log\left(n_1 q^{i-1}\right))$$

.

Выражение в скобках – время обновления *i*-ого *kd*-дерева, а выражение перед скобками – частота его обновления. Упрощая выражение, получим:

$$mq \sum_{i=1}^{p} \log\left(n_1 q^{i-1}\right) = mq \sum_{i=1}^{p} (\log(n_1) + (i-1)log(q))$$
$$= m * q * p * \log(n_1) + m * q * \frac{1}{2} p(p-1)\log(q) = T_{modify}(n_1, q);$$

Результирующее время обновления мульти *kd*-дерева составляет

$$O(q * n_1 \log(q * n_1) + m * q * p * \log(n_1) + m * q * \frac{1}{2} p(p-1)\log(q))$$

Значение *p* можно получить из соотношение (3).

Оценка наиболее эффективных параметров $n_1$ и $q$ производилась следующим образом. Вначале требовалось определить соотношение коэффициентов в оценке времени добавления и поиска. Для этого использовались результаты, полученные при работе одного *kd*-дерева. На поиск s = 526 точек было затрачено *ts* = 79 мс, а на добавление *m* = 678 точек было затрачено *tm* = 750 мс, число точек в дереве составляло *n* = 44828.

Для построения графика зависимости времени работы алгоритма от различных параметров, требуется знать не только асимптотику времени работы алгоритма, но и коэффициенты отражающие трудоемкость операций алгоритма. Для операции поиска элементов в *kd*-дереве это коэффициент $C1$, для операции построения *kd*-дерева это коэффициент $C2$.

*C1*s*log(n)=ts;*

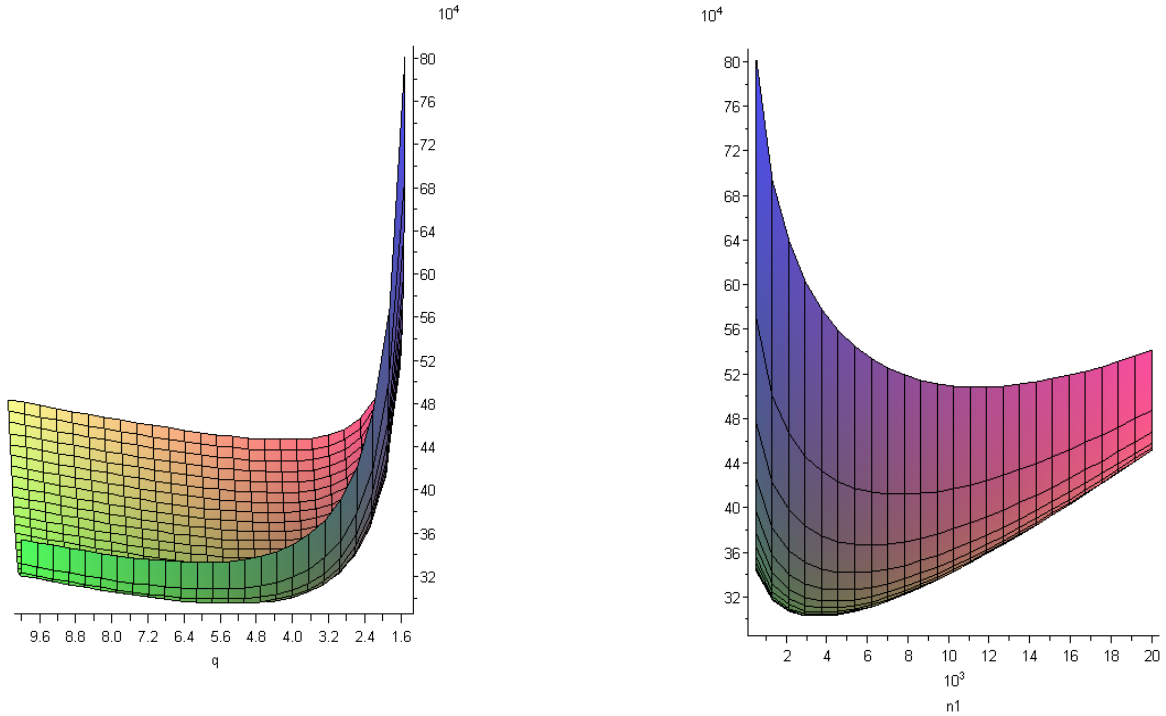
*C2*n*log(n)=tm;*

Из практических данных приведенных выше следует, что *C = C1/C2* примерно равно девяти. Эта константа и применялась для вычисления оптимальных параметров мульти *kd*-дерева.

В качестве функции для минимизации было выбрано соотношение:

$$f(n_1, q) = T_{modify}(n_1, q) + C * s * T_{search}(n1, q)$$

Минимизация осуществлялась при следующих параметрах: число точек в структуре *n* = 50000, число запросов на поиск *s* = 500, число добавляемых точек *m* = 500. Использовался графический метод. График приведен ниже (рис. 13, 14) в разных ракурсах.




а Б

Рис. 13. Разные ракурсы: вид вдоль оси *n1* (а), вид вдоль оси *q* (б)

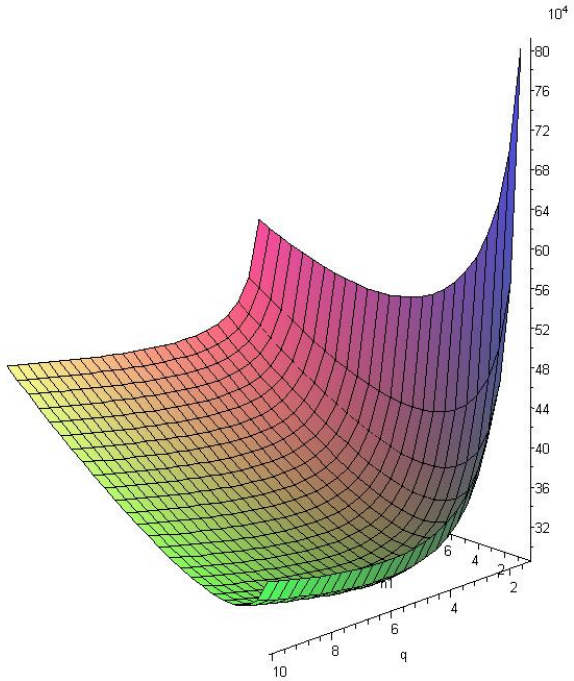



Рис. 14. Вид графика

Из графика (рис. 13, 14) видно, что минимум достигается при $q$ = 5,5 и $n_1$=3,5. Данные параметры предлагается применять при числе точек в мульти *kd*-дереве не превышающем 50000. Все вычисления выполнялись в *Maple 12.0*, текст программы приведен в Приложении.

### 4.2. Сравнение времени работы контейнеров для *k*-точек в реальной ситуации

При тестировании структур для хранения ориентиров было рассмотрено два теста. Первый тест – движение по замкнутой траектории диаметром порядка четырех метров при доступном для наблюдения пространстве размера 6 * 8 м. Второй – движение в замкнутом помещении размера 10 * 12 м, длина траектории – порядка 80 м. При этом в первом тесте точки считались базовыми только после того, как встретились не менее трех раз за семь итераций, а во втором – все точки сразу считались базовыми.

Результаты сравнения приведены в таблицах. В табл. 1 – результаты первого теста, в табл. 2 – второго.

Таблица 1. Сравнительные характеристики быстродействия структур хранения *k*-точек

| Структура | *kd*-дерево | Мульти *kd*-дерево |
|---|---|---|
| Среднее время добавления, мс | 85 | 25 |
| Максимальное время добавления, мс | 204 | 141 |
| Среднее время поиска, мс | 70 | 174 |
| Максимальное время поиска, мс | 157 | 375 |
| Среднее время одной итерации, мс | 155 | 200 |
| Максимальное время одной итерации, мс | <360 | <515 |

.

Таблица 2. Сравнительные характеристики быстродействия структур хранения *k*-точек

| Структура | *kd*-дерево | Мульти *kd*-дерево |
|---|---|---|
| Среднее время добавления, мс | 524 | 33 |
| Максимальное время добавления, мс | 1219 | 609 |
| Среднее время поиска, мс | 55 | 164 |
| Максимальное время поиска, мс | 110 | 406 |
| Среднее время одной итерации, мс | 579 | 199 |
| Максимальное время одной итерации, мс | <1329 | <1015 |

В первом тесте максимальное число *k*-точек в структуре составило 13000, а во втором тесте – 65000. Число поисковых запросов на одном шаге в среднем составляло порядка 400.

Результаты первого теста свидетельствуют о том, что выигрыш в скорости добавления точек от применения мульти *kd*-дерева меньше проигрыша из-за замедления поиска по нескольким *kd*-деревьям. Таким образом, при реализации алгоритма *SLAM* в пространствах, число точек-ориентиров в которых невелико (не превосходит 20000), эффективнее использовать одно *kd*-дерево.

Применение одного *kd*-дерева позволяет осуществлять быстрый поиск точек, тогда как время на его перестроение в данном случае еще мало.

Ниже приведены результаты сравнения структур на большем тесте. Результаты качественно отличаются от первого случая. Время на перестроения одного *kd*-дерева растет как $O(n * \log n)$. Поэтому с ростом числа точек в нем производительность существенно падает. Применение мульти *kd*-дерева дает существенный эффект. Среднее время добавления мало и примерно совпадает со средним временем обновления *kd*-дерева размера 3000. Время работы мульти *kd*-дерева в худшем слу-

чае достаточно велико, но меньше, чем у *kd*-дерева. Плохим случаем является перестроение сразу всех хранимых *kd*-деревьев

### 4.3. Выводы по главе 4

По итогам тестирования структур для хранения пространственных ориентиров следует сделать следующие выводы.

1. При большом числе поисковых запросов и небольшом числе пространственных ориентиров (не более 20000) возможно (и эффективно) применение одного не модифицируемого *kd*-дерева.
2. При небольшом числе поисковых запросов или большом числе пространственных ориентиров становится эффективным применение структуры мульти *kd*-дерево. Она обеспечивает несколько более медленный поиск по сравнению с *kd*-деревом, но зато высокую среднюю скорость обновления и, как следствие, небольшое среднее время работы одной итерации алгоритма.
3. Следует проводить оценку оптимальных параметров мульти *kd*-дерева применительно к реальной ситуации – ожидаемому числу поисковых запросов, добавляемых *k*-точек, максимальному числу *k*-точек в контейнере.

# ЗАКЛЮЧЕНИЕ

В данной работе рассматривалась задача *stereo SLAM*. Был реализован алгоритм совместного определения местоположения и построения карты пространственных ориентиров. Алгоритм может работать в режиме реального времени при небольших относительных скоростях движения робота, частота обработки изображений составляет порядка два – три герца. Точность определения углов поворота составляет порядка 10% при одновременном движении и повороте.

Улучшение характеристик алгоритма возможно, в первую очередь, за счет реализации ансцентного фильтра Калмана для местоположения и ориентации робота. Также важна реализация уточнения 3D-координат точек-ориентиров при замыкании цикла.

В данной работе был предложен метод адаптации не модифицируемого *kd*-дерева к задаче стерео *SLAM*. Данная адаптация позволяет повысить производительность и улучшить асимптотику быстродействия программы. Также произведено сравнение полученной структуры с не модифицируемым *kd*-деревом, теоретически определена асимптотика времени работы, проанализирована ее эффективность в реальной ситуации, определены оптимальные параметры для характерного числа пространственных ориентиров.

## ИСТОЧНИКИ


1. *SLAM* in realistic environments. http://www.nada.kth.se/utbildning/forsk.utb/avhandlingar/lic/020220.pdf
2. *Riisgaard S., Blas M. R. SLAM* for Dummies. http://ocw.mit.edu/NR/rdonlyres/Aeronautics-and-Astronautics/16-412JSpring-2005/9D8DB59F-24EC-4B75-BA7A-F0916BAB2440/0/1aslam_blas_repo.pdf
3. *Smith R., Self M., Cheeseman P.* Estimating Uncertain Spatial Relationships in Robotics. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.7569&rep=rep1&type=pdf
4. *Thrun S., Montemerlo M., Koller D., Wegbreit B., Nieto J., Nebot E.* FastSLAM: an efficient solution to the simultaneous localization and mapping problem with unknown data association. http://robots.stanford.edu/papers/Thrun03g.pdf
5. *Infinite impulse response*. http://www.bores.com/courses/intro/iir/index.htm
6. *Welch G., Bishop G.* An Introduction to the Kalman Filter. http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf
7. *Riberio M. I.* Kalman and Extended Kalman Filters: Concept, Derivation and Properties. 2004. http://users.isr.ist.utl.pt/~mir/pub/kalman.pdf
8. *Van der Merve R., Wan E. A.* The sqare-root unscented Kalman filter for state and parameter-estimation. http://speech.bme.ogi.edu/publications/ps/merwe01a.pdf
9. *Bries M., Maskell S. R., Wright R. A* Rao-Blackwellised unscented Kalman filter. http://www-sigproc.eng.cam.ac.uk/~mb511/papers/rbukf.pdf
10. *Naerum E., King H. H., Hannaford B.* Robusness of unscented Kalman filter for state and parameter estimation in an elastic transmission. http://www.roboticsproceedings.org/rss05/p25.pdf
11. *Numerical recipes in Fortran 77*: the art if scientific computing. http://www.mpi-hd.mpg.de/astrophysik/HEA/internal/Numerical_Recipes/f2-9.pdf
12. *Van der Merwe R., Wan E. A.* The Unscented Kalman Filter for Nonlinear Estimation. http://www.lara.unb.br/~gaborges/disciplinas/efe/papers/wan2000.pdf

13. *Durrant-Whyte L.* Mobile robot localization by tracking geometric beacons. http://www.cs.cmu.edu/~biorobotics/papers/sbp_papers/l/leonard_ieeetroa19.pdf
14. *Se S., Lowe D., Little J.* Vision-based mobile robot localization and mapping using scale-invariant features. http://www.cs.ubc.ca/~lowe/papers/icra01.pdf
15. *Lowe D. G.* Distinctive image features from scale-invariant keypoints. 2004. http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf
16. *Lowe D. G.* Object recognition from local scale-invariant features. http://robotics.caltech.edu/readinggroup/vision/LoweICCV99.pdf
17. *Meng Y., Dr. Tiddeman B.* Implementing the scale invariant feature transform (SIFT) method. http://www.cs.st-andrews.ac.uk/~yumeng/yumeng-SIFTreport-5.18_bpt.pdf
18. *Bay H., Tuytelaars T., Van Gool L.* SURF: speeded-up robust features. http://www.vision.ee.ethz.ch/~surf/eccv06.pdf
19. *Terryberry T. B., French L. M., Helmsen J.* GPU accelerating speeded-up robust features. http://people.xiph.org/~tterribe/pubs/gpusurf.pdf
20. Difference of Gaussian scale-space pyramids for SIFT feature detection. http://www.ballardblair.com/projects/Difference_of_Gaussian_paper.pdf
21. *Evans C.* Notes on the OpenSURF library. 2009. http://opensurf1.googlecode.com/files/OpenSURF.pdf
22. *Bentley J.L.* Multidimensional binary search trees for associative searching. 1975. http://www.cs.bgu.ac.il/~atdb082/wiki.files/paper4.pdf
23. *Beis J.S., Lowe D.G.* Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. http://www.cs.ubc.ca/~lowe/papers/cvpr97.pdf
24. *Dailey M.N., Parnichkun M.* Landmark-based simultaneous localization and mapping with stereo vision. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=8380BA4A727E38F6838DD7ACB0BF09B3?doi=10.1.1.59.1771&rep=rep1&type=pdf

25. *Marks K. T., Howard A., Bajracharya M., Cottrell G. W., Matthies L.* Gamma-SLAM: using stereo vision and variance grid maps for SLAM in unstructured environments. http://cseweb.ucsd.edu/~gary/pubs/slam_icra08-FINAL.pdf
26. *Sarkka S., Vehtari A., Lampinen J.* Rao-Blackwellized particle filter for multiple target tracking. http://www.lce.hut.fi/~ssarkka/pub/nrbmcda-preprint.pdf
27. *Grisetti G., Stachniss C., Burgard W.* Improved techniques for grid mapping with Rao-Blackwellized particle filters. http://www.informatik.uni-freiburg.de/~stachnis/pdf/grisetti06tro.pdf
28. *Jackel L.*DARPA`s LAGR and UPI programs. http://www.laas.fr/IFIPWG/Workshops&Meetings/49/workshop/06%20jackel.pdf
29. *Eliazar A.I., Parr R.* DP-SLAM 2.0. http://www.cs.duke.edu/~parr/dpslam2.pdf
30. *Lorsakul A., Suthakorn J., Sinthanayothin C.* Point-cloud-to-point-cloud technique on tool calibration for dental implant surgical path tracking. http://www.bartlab.org/Dr.%20Jackrit%60s%20Papers/ney/4.Lorsakul_ProcSPIE.pdf
31. *Zuliani M.* RANSAC for dummies. http://vision.ece.ucsb.edu/~zuliani/Research/RANSAC/docs/RANSAC4Dummies.pdf
32. Колесный робот Иван. http://transas.com/vision/ai/robot/

## ПРИЛОЖЕНИЕ

Программа вычисления оптимальных коэффициентов мульти *kd*-дерева. Математический пакет *Maple 12*.

Оценка времени поиска в мульти *kd*-дереве:

$$searchKd := p\$\log_2(n1)\,C\,\frac{p\$(p K\ 1)}{2}\$\log_2(q);$$

Оценка среднего времени добавления в мульти *kd*-дерево:

$$addKd := n1\cdot\log_2(n1) + m\cdot q\cdot p\cdot\log_2(n1) + \frac{m\cdot q}{2}\cdot p\cdot(p-1)\cdot\log_2(q);$$

Оценка среднего времени одной итерации поиска и добавления мульти kd-дерева:

$$iterTime := addKd\,C\ C\$s\$searchKd;$$

Число точек в *kd*-дереве:

$$n := 50000;$$

Константа, характеризующая отношение быстродействия поиска и добавления в реальной ситуации:

$$C := 9;$$

Число запросов на поиск:

$$s := 500;$$

Число добавляемых на одной итерации *k*-точек:

$$m := 500;$$

Отображение среднего времени работы одной итерации мульти *kd*-дерева для параметров: максимальный размер первого *kd*-дерева от 500 до 20000, коэффициент увеличения максимального размера каждого следующего *kd*-дерева от 1,5 до 10:

$$plot3d(iterTime,\ n1 = 500..20000,\ q = 1.5..10,\ axes = normal).$$