

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

**МОДЕЛИРОВАНИЕ И ВИЗУАЛИЗАЦИЯ РАЗЛИЧНЫХ СТРАТЕГИЙ SLAM С ПОМОЩЬЮ LIDAR В ПОМЕЩЕНИЯХ ДЛЯ МОБИЛЬНЫХ РОБОТОВ**

Работу выполнил К. В. Осанов

#### РЕФЕРАТ

Выпускная квалификационная работа 51 с., 8 ч., 27 рис., 14 источников.

РОБОТОТЕХНИКА, SLAM, ФИЛЬТР КАЛМАНА, LIDAR, АВТОНОМНОСТЬ.

Целью данной работы является разработка программы, которая моделирует и визуализирует поведение робота в различных закрытых пространствах, с помощью данных полученных от LIDAR строит карту местности, и локализует робота.

В процессе написания программы были рассмотрены различные стратегии SLAM, датчики, которые используются в решении данной проблемы.

В результате выпускной квалификационной работы была создана программа, которая позволяет визуализировать и моделировать поведение робота, строить карту с помощью данных полученных от LIDAR. При

реализации использовались Visual Studio 2017

Содержание

[ВВЕДЕНИЕ 6](#_Toc196655538)

[1 Обзор предметной области 8](#_Toc196655539)

[1.1 Роботы 8](#_Toc196655540)

[1.2 Датчики 9](#_Toc196655541)

[1.2.1 Дальномеры 10](#_Toc196655542)

[1.2.2 Одометры 11](#_Toc196655543)

[1.2.3 Видео камеры 12](#_Toc196655544)

[1.3 Модель окружающей среды 12](#_Toc196655545)

[1.4 Задача SLAM 13](#_Toc196655546)

[1.4.1 Фильтр Калмана 16](#_Toc196655547)

[1.4.2 Расширенный фильтр Калмана 20](#_Toc196655548)

[1.4.3 Частичный фильтр SLAM 22](#_Toc196655549)

[2 Известные реализации 23](#_Toc196655550)

[2.1 FastSLAM 24](#_Toc196655551)

[2.2 Graph-Based SLAM 25](#_Toc196655552)

[2.3 Gamma-SLAM 26](#_Toc196655553)

[2.4 Иные реализации SLAM 28](#_Toc196655554)

[3 Реализованный метод 29](#_Toc196655555)

[3.1 Постановка задачи 30](#_Toc196655556)

[3.2 Реализация 30](#_Toc196655557)

[3.2.1 Перемещение робота 30](#_Toc196655558)

[3.2.2 LIDAR 31](#_Toc196655559)

[3.2.3 Моделирование работы робота 34](#_Toc196655560)

[3.2.4 Построение карты 37](#_Toc196655561)

[3.2.5 Автономное движение робота 38](#_Toc196655562)

[3.2.6 Самолокализация 42](#_Toc196655563)

[3.3 Результаты работы программы 45](#_Toc196655564)

[ЗАКЛЮЧЕНИЕ 51](#_Toc196655565)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 51](#_Toc196655566)

# ВВЕДЕНИЕ

Робототехника позволяет автоматизировать технические системы и развитие навыков в этой сфере поможет облегчить многие задачи. Одной из задач робототехники, является задача создания различного вида роботов. Навигация роботов может происходит в автономном режиме. Поэтому возникают различные вопросы, такие как: где находится робот, куда ему двигаться, как выбрать путь чтобы достичь того или иного положения. Однако, сложно реализовать робота, который бы был полностью автономным и мог преодолевать любые препятствия, а также строить карту пройденной местности. Или же в случае возникновения непредвиденной ситуации, например, возникновение нового препятствия, была возможность распознать его и продолжить свою деятельность.

Изучение данной проблемы и возможно поиск новых решений позволит использовать различного вида роботов там, где это еще не используется. Ранее решение этой проблемы было трудоемким и дорогим, так как требовались точные датчики и ресурсозатратные вычисления. На данный момент для этого можно совмещать различные датчики, с помощью которых можно узнать расстояние до объекта, использовать видеокамеры для фиксирования неких ориентиров, однако можно воспользоваться одометрами, которые позволят узнать на сколько градусов повернулся робот или же какой путь он проехал. Также появляются различные новые методы для реализации данной проблемы.

Использование программ для моделирования роботов позволяет производить испытания на стадии проектирования робота с целью выбора датчиков для его оснащения и с учетом особенностей среды в которой он будет работать.

Simultaneous localization and mapping – синхронное определение местоположения и составление карты [1]. Задача SLAM связана с построением карты неизвестного пространства мобильным роботом во время навигации по строящейся карте.

SLAM имеет такие задачи как нахождение ориентиров, вычисление местоположения, построение карты. Существует много решений, что позволяет реализовывать различные комбинаций, например, если окружающая среда является открытой местностью, то необходимы уже некоторые изменения.

Целью данной работы является разработка программы, которая моделирует и визуализирует поведение робота в различных закрытых пространствах, с помощью данных полученных от LIDAR строит карту местности, и локализует робота.

# 1 Обзор предметной области

## 1.1 Роботы

Робот – приводной механизм, программируемый по двум и более осям, имеющий некоторую степень автономности, движущийся внутри своей рабочей среды и выполняющий задачи по предназначению [2].

Автономность – способность выполнять задачи по предназначению, основанная на текущем состоянии изделия и особенностях считывания данных без вмешательства человека [2].

Полностью автономный робот должен иметь следующие способности:

* получать и самостоятельно обрабатывает информацию об окружающей среде;
* работать больший период времени без человеческого вмешательства;
* перемещаться целиком или перемещать какую-то свою часть в пространстве без человеческой помощи;
* избегать создания опасных ситуаций для человека, имущества или для самого себя – если только это не предусмотрено спецификацией, характеристиками робота или поставленными ему задачами.

Мобильный робот – робот с автономным управлением, который может самостоятельно передвигаться [2].

Мобильных роботов можно классифицировать по месту обитания:

* наземные или домашние роботы, обычно называют беспилотными наземными транспортными средствами;
* роботы доставки и транспортировки могут перемещать материалы и материалы через рабочую среду;
* воздушные роботы, обычно называются беспилотными летательными аппаратами;  подводные роботы, обычно называют автономными подводными аппаратами;
* полярные роботы, предназначенные для навигации в ледяных средах, заполненных трещинами.

Классификация по способу перемещения роботов: колесный робот, шагающий робот, двуногий робот, гусеничный робот, рельсовый робот, антропоморфный робот.

Локализация – распознавание расположения мобильного робота или его идентификация на карте окружающей среды [2].

Ориентир – искусственный или естественный объект, различимый на карте окружающей среды, используемый для локализации мобильного робота [2].

Препятствие – статический или движущийся объект, находящийся в любой плоскости, препятствующий заданному движению [2].

Картографирование или построение карты – разработка карты окружающей среды для описания окружающей среды с ее геометрическими и различимыми особенностями, ориентирами и препятствиями [2].

Навигация – принятие решения и управление направлением движения на основе локализации объекта и использования карты окружающей среды [2]. Также навигация может включать планирование пути для перемещения из одного расположения в другое и полное покрытие области.

## 1.2 Датчики

Датчик робота – преобразователь, используемый для получения внутренней и внешней информации для управления роботом [3]. Реализация SLAM невозможна без применения датчиков, которые дают информацию об окружающей среде.

Можно выделить три группы датчиков.

1. Дальномеры.
2. Одометры.
3. Видео камеры.

Также существую различные датчики, которые могут быть использованы не только для навигации, но и для других целей, например, датчик дыма, газа, движения и так далее. В рамках данной работы необходимы датчики только для навигации робота.

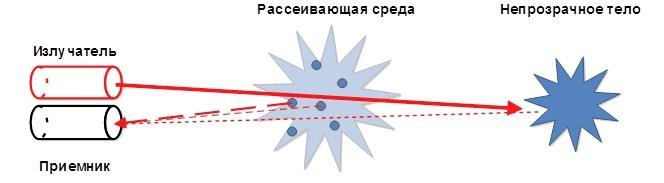
### 1.2.1 Дальномеры

Дальномер – устройство, предназначенное для определения расстояния от наблюдателя до объекта. Рассмотрим такие дальномеры как LIDAR и

Sonar.

LIDAR – технология получения и обработки информации об удалѐнных объектах с помощью активных оптических систем, использующих явления отражения света и его рассеяния в прозрачных и полупрозрачных средах [3]. Лазерные 2D-сканеры позволяют получать расстояние до препятствий одновременно для большого угла обзора. Получаемые данные очень точные, и легки в обработки. Но такие сканеры имеют достаточно высокую цену, а также возникают проблемы с их использование в среде где есть стекло или вода, из-за отражения и рассеивания соответственно.

На рисунке 1 показан принцип действия LIDAR. Направленный луч источника излучения отражается от целей, возвращается к источнику и улавливается высокочувствительным приѐмником, то есть светочувствительным полупроводниковым прибором, и время отклика прямо пропорционально расстоянию до цели.



#### Рисунок 1 – Принцип действия LIDAR

Вторым способом получения расстояние до препятствий является использование сонаров. Ультразвуковые датчики основаны на измерении свойств акустических волн с частотами примерно в 40 кГц. Они обычно работают путем генерации высокочастотного звукового импульса, а затем приема и оценки свойств эхо - импульса. Их стоимость гораздо меньше чем у лазерных сканеров, но при этом точность измерений не столь высока и часто приходят неточные данные. Зато использование сонара под водой гораздо предпочтительней.

Результаты, полученные с помощью сонаров, будут приблизительно такие же, как и от LIDAR, но робот должен будет повернуться вокруг своей оси и на каждом градусе сделать измерения, что повлечет за собой множество ошибок.

### 1.2.2 Одометры

Одометрия – использование данных о движении приводов, для оценки перемещения [4]. Стандартной схемой одометрии робота, является использование энкодеров, считывающих угол поворота колѐс. Энкодер – устройство преобразующее линейное или угловое перемещение в последовательность сигналов, позволяющих определить величину перемещения. Одной из проблем использования одометров является пробуксовка или скольжение колес. Это возникает из-за резкого торможения, неоднородного покрытия или при встрече с препятствием. Поэтому использовать энкодеры лучше в качестве дополнение к другим датчикам.

### 1.2.3 Видео камеры

Использование видео камер может приводить к ошибкам из-за изменения уровня освещенности, например, в условиях отсутствия света очень сложно производить вычисления. Для произведения вычислений данных от видеокамер требуются большие вычислительные ресурсы. Также возникает проблема при однородной окружающей среде, отражающихся поверхностях. Однако, использование стереосистем дает информацию о расстоянии до объекта и позволяет выявлять много дополнительных деталей.

## 1.3 Модель окружающей среды

Под моделью окружающей среды понимается карта или модель, описывающая окружающую среду и ее основные характеристики.

Можно выделить два типа окружающей среды такие как закрытая и открытая. Все что находится в комнате, здании, либо еще каком-то строении понимается как закрытая окружающая среда. Все что находится вне, то есть на открытом воздухе, понимается как открытая окружающая среда.

Среда может быть статической и динамической. Статической

называется та среда, в которой препятствия не изменяют своего положения, в течении того, как робот производит свою деятельность.

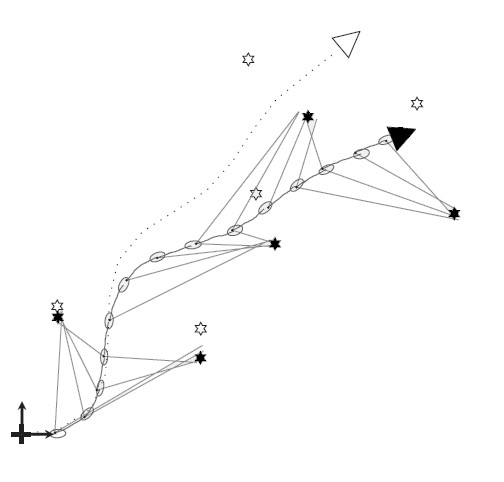
Динамическая среда – это среда объекты в которой могут изменить свое положение, в то время пока робот находится в ней.

Таким образом можно классифицировать различные виды окружающей робота среды. В зависимости от того, в какой среде находится робот необходимо выбирать те или иные способы его передвижения, различные способы обработки данных, так как различные реализации, могут быть успешно применены, например, в закрытой среде, но попав в открытую, результат будет хуже.

## 1.4 Задача SLAM

Simultaneous localization and mapping – синхронное определение местоположения и составление карты. Задача SLAM связана с построением карты неизвестного пространства мобильным роботом во время навигации по строящейся карте. Исходными данными является некоторая среда, например, какое-то помещение, или открытая местность, возможно водная или воздушная среда, и робот с датчиками. Необходимо знать местоположение робота в каждый момент времени, данные о карте, которые были доступны с стартовой позиции робота, до текущей позиции включительно. Местоположение робота будет получено в системе локальных координат, связанных с начальным положением робота, так как изначальной информации о положении относительно пространства нет. Главным требованием к построению карты является возможность робота ориентироваться по ней, так как без вычисления местоположения, построение карты будет осуществляться независимо, а это приведет к накоплению ошибок, на каждом шаге, и приведет к росту погрешности.

На рисунке 2 показан пример задачи. Тут показаны две траектории, одна из которых реальная траектория робота, которая изображена пунктирной линией и вычисленная алгоритмом траектория, эта линия соединяет эллипсы. Промежуточные позиции робота указаны эллипсами. Реальные препятствия или некие ориентиры изображены полыми звездами, а полученные с помощью алгоритма закрашенными звездами. На рисунке видно, что на каждом шаге появляется погрешность. Полученная карта является картой ориентиров [5].



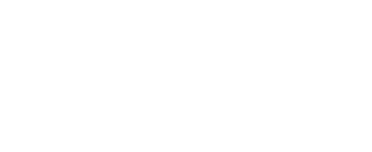
#### Рисунок 2 – Пример SLAM

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. В задачи SLAM как правило, выделяются основные подзадачи:

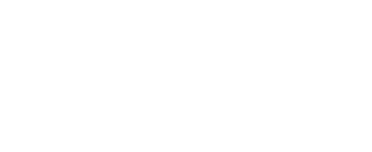
* рекурсивный фильтр;
* нахождение ориентиров в пространстве;
* поиск соответствий между ориентирами;
* пересчет положения робота;
* уточнение положения ориентиров на карте.

При большом разнообразии методов, применяемых в задаче синхронного определения местоположения и построения карты, можно выделить некоторую классическую, весьма общую схему работы алгоритма [1].

Пример стандартной схемы алгоритмов SLAM показана на рисунке 3.

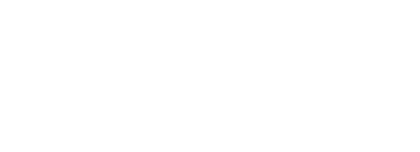


Поиск ориентиров



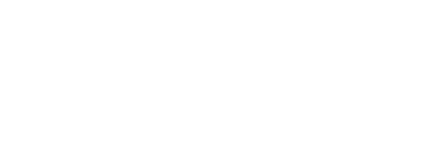
Поиск

соответствий



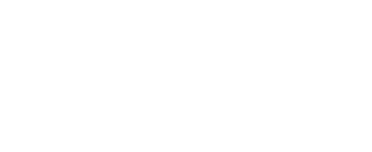
Фильтр

Уточнение положения



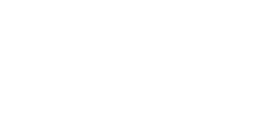
Фильтр

Повторное наблюдение



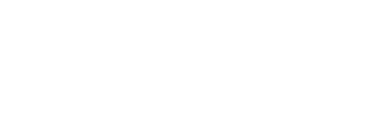
Фильтр

Новые ориентиры



Данные от

дальномеров



Вычисление

положения

#### Рисунок 3 – Схема работы SLAM

На каждом шаге алгоритма на вход подаются данные от различных видов дальномеров. Используя эти данные находятся различные ориентиры в пространстве и определяются разные описания, которые нужны для поиска соответствий. В ходе работы алгоритма заполняется структура, которая хранит эти ориентиры и их описания. Во время поиска соответствий для каждых обнаруженных ориентиров ищутся соответствия в этой структуре. Если соответствия не найдены, то происходит добавление ориентира в эту структуру. Если же соответствие было обнаружено, то текущий ориентир будет использоваться для вычисления текущего положения.

После того как вычисление положения было выполнено, применяется рекурсивный фильтр для того чтобы снизить шум в вычисленном положении. После этого, для обнаруженных ориентиров, у которых нашлись соответствия, происходит уточнение их положения с помощью рекурсивного фильтра.

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.

Большинство алгоритмов SLAM основывается на трех различных подходах:

* расширенный фильтр Калмана для SLAM;
* частичный фильтр SLAM;
* основанный на графах SLAM.

### 1.4.1 Фильтр Калмана

Фильтр Калмана базируется на дискретных по времени линейных динамических системах [6]. Движение автономных роботов плохо аппроксимируется линейным движением поэтому фильтры Калмана не получили широкого распространения в реализациях задачи SLAM. Но в то же время можно применять 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 , матрицу наблюдений , ковариационную матрицу процесса , ковариационную матрицу шума измерений , при наличии управляющих воздействий – матрицу их коэффициентов .

Модель системы подразумевает, что истинное состояние в момент k получается из истинного состояния в момент k-1 в соответствии с уравнением

, (1)

где  – матрица эволюции процесса, которая воздействует

на вектор ;

 – матрица управления, которая прикладывается к вектору управляющих воздействий ;

 – нормальный случайный процесс с нулевым математическим ожиданием и ковариационной матрицей , который описывает случайный характер эволюции системы, .

В момент k производится измерение  истинного вектора состояния

, которые связаны между собой уравнением

, (2)

где  – матрица измерений, связывающая истинный вектор состояния

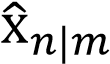
и вектор произведенных измерений;

 – белый гауссовский шум измерений с нулевым математическим ожиданием и ковариационной матрицей , .

Начальное состояние и векторы случайных процессов на каждом такте  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 ему необходима оценка состояния на предыдущем такте работы и измерения на текущем такте. Данное свойство отличает его от пакетных фильтров, требующих в текущий такт работы знание истории измерений или оценок.

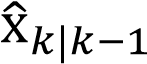
Далее под записью  будем понимать оценку истинного вектора  в момент n с учетом измерений с момента начала работы и по момент m включительно.

Состояние фильтра задается двумя переменными:

*  – апостериорная оценка состояния объекта в момент k

полученная по результатам наблюдений вплоть до момента k включительно;

*  – 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.

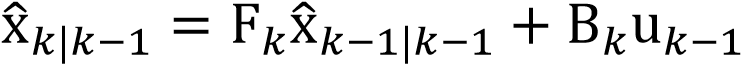
Итерации фильтра Калмана делятся на две фазы: экстраполяция и коррекция. Во время экстраполяции фильтр получает предварительную оценку состояния системы  на текущий шаг, по итоговой оценке, состояния с предыдущего шага. Эту предварительную оценку также называют априорной оценкой состояния, так как для еѐ получения не используются наблюдения соответствующего шага. В фазе коррекции априорная экстраполяция дополняется соответствующими текущими измерениями для коррекции оценки. Скорректированная оценка называется апостериорной оценкой состояния, либо просто оценкой вектора состояния

. Обычно эти две фазы чередуются: экстраполяция производится по результатам коррекции до следующего наблюдения, а коррекция производится совместно с доступными 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. Аналогично, если независимые измерения доступны только в отдельные такты работы, всѐ равно возможны коррекции.

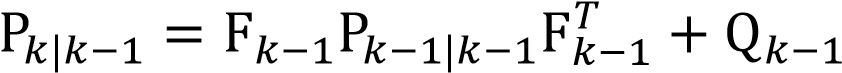
Далее рассмотрим работу классического оптимального фильтра Калмана.

Этап экстраполяции

Экстраполяция вектора состояние системы по оценке вектора состояния и примененному вектору управления с шага k-1 на шаг k

. (3)

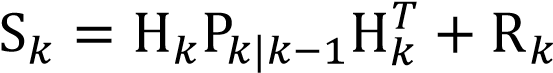
Ковариационная матрица для экстраполированного вектора состояния

. (4)

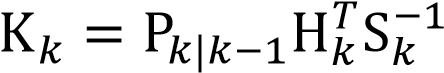
Этап коррекции

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

. (5) Ковариационная матрица для вектора отклонения

. (6)

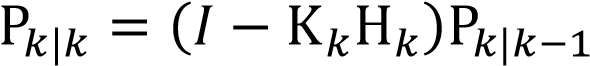
Оптимальная по Калману матрица коэффициентов усиления, формирующаяся на основании ковариационных матриц, имеющейся экстраполяции вектора состояния и полученных измерений

. (7)

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

. (8)

Расчет ковариационной матрицы оценки вектора состояния системы

. (9)

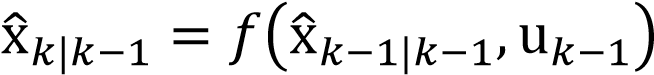
Выражение для ковариационной матрицы оценки вектора состояния системы справедливо только при использовании приведенного оптимального вектора коэффициентов. В общем случае это выражение имеет более сложный вид.

### 1.4.2 Расширенный фильтр Калмана

Расширенный фильтр Калмана является нелинейной версией фильтра Калмана [7]. Основная идея, применяемая в расширенном фильтре Калмана, состоит в приближении функций состояния и наблюдения с использованием их первых производных.

Этап экстраполяции

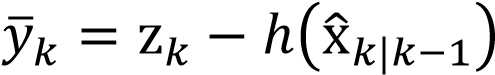
Экстраполяция вектора состояние системы по оценке вектора состояния и применимому вектору управления с шага k-1 на шаг k:

. (10)

Ковариационная матрица для экстраполированного вектора состояний является формулой (4).

Этап коррекции

Отклонения полученного на шаге k наблюдения от наблюдения, ожидаемого при произведенной экстраполяции

. (11)

Ковариационная матрица для вектора отклонения обозначается формулой (6).

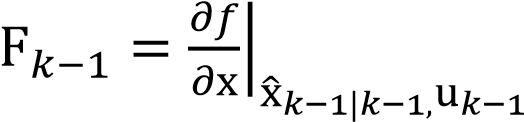
Оптимальная по Калману матрица коэффициентов усиления, формирующаяся на основании ковариационных матриц, имеющейся экстраполяции вектора состояния и полученных измерений обозначается формулой (7).

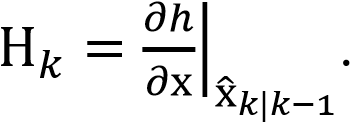
Коррекция ранее полученной экстраполяции вектора состояния – получение оценки вектора состояния системы обозначается формулой (8).

Расчет ковариационной матрицы оценки вектора состояния системы обозначается формулой (9).

Матрицы изменения состояния системы и наблюдения определяются

Якобианами:

, (12)

 (13)

Расширенный фильтр Калмана в общем случае не совсем оптимален, в отличии от линейного аналога. Из-за ошибочных вычислений или же некорректного моделирования процесса результаты могут быстро расходиться. Также вычисляемая матрица ковариации склонна к недооценке реальной ковариации поэтому риски становятся мало связанными с реальной ситуацией.

### 1.4.3 Частичный фильтр SLAM

Частичный фильтр – рекурсивный алгоритм для численного решения проблем оценивания, фильтрации, сглаживания. особенно для нелинейных и не-гауссовских случаев [8]. В сравнении с расширенным фильтром Калмана частичные фильтры не зависят от методов линеаризации или аппроксимации. Обычный расширенный фильтр Калмана плохо справляется с существенно нелинейными моделями, а также в случае шумов системы и измерений, сильно отличающихся от гауссовых. А также частичные фильтры более требовательны к вычислительным ресурсам.

Частичный фильтр предназначен для оценки последовательности скрытых переменных  для , на основании наблюдений  при . Для простоты изложения будем считать, что рассматривается динамическая система, и  и  – действительные вектора состояния и измерений соответственно.

Стохастическое уравнение состояния системы имеет вид

, (14)

где  – функция изменения состояния системы;

– случайная величина, возмущающее воздействие.

Уравнение измерений

, (15)

где  – функция измерения;

случайная величина, шум измерений.

Функции  и  в общем случае нелинейные, а статистические характеристики шума системы и измерений предполагаются известными.

Задачей фильтрации является получение оценки  на основе известных к моменту  результатов измерений .

# 2 Известные реализации

Существуют различные реализации SLAM, применяемые как для реальных роботов, находящихся в условиях реальной среды, так и смоделированных с помощью компьютера. При применении SLAM на

практике в основном используют высокоточные одометры, лазерные сканеры или стереопары с высокой точностью калибровки. Подобная аппаратура позволяет точно вычислять все препятствия или более лучшее обнаружение маяков. В то же время, стоимость роботов на которых установлены такие датчики, достаточно высока.

## 2.1 FastSLAM

FastSLAM – один из подходов решения задач SLAM. В основе

алгоритма лежит так называемый фильтр частиц и применение Байесовской сети [5]. В FastSLAM одна большая карта рассматривается как совокупность локальных подкарт, что позволяет убрать зависимость ориентиров друг от друга и таким образом значительно сократить время пересчета оценки состояния системы. Одним из преимуществ использования FastSLAM является то, что сложность алгоритма вычисления упрощается. Это позволяет, в свою очередь, решить проблему полиномиальной сложности алгоритма на основе расширенного фильтра Калмана при решении задач SLAM и избежать ее в FastSLAM. Но потенциальное падение точности, связанное с игнорированием корреляции ошибок оценки положений ориентиров, является минусом данного метода. FastSLAM был изначально разработан для использования лазерных сканеров и построения 2D карт, но на практике этот метод адаптировали и для трехмерного использования, и использование камер в качестве датчиков.

На рисунке 4 показан физический робот, который создан при финансировании NASA для Mars Rover. Робот Pioneer, оснащен лазерным дальномером SICK.



#### Рисунок 4 – Пример работы FastSLAM

Робот проехал по примерной прямой, генерируя карту. Масштабируя свойства подхода, оценили карту и ошибки в зависимости от количества ориентиров, и число частиц, соответственно. Увеличение числа ориентиров уменьшает количество ошибок в карте и позиционировании робота. А также, чем больше ориентиров, тем меньше ошибок. Увеличение числа частиц оказывает положительное влияние на карту и сокращает появление ошибок.

Алгоритм FastSLAM был тщательно протестирован в рамках различных условия. Эксперименты в реальном мире были дополнены cистематическими симуляционными экспериментами и исследованиями масштабирования подхода. В целом, результаты показывают благоприятное масштабирование для большого количества ориентиров и небольших частиц.

## 2.2 Graph-Based SLAM

Graph-Based SLAM создает упрощенную задачу оценки путем абстрагирования необработанных измерений датчиков [9]. Эти необработанные измерения заменяются ребрами в графе, которые затем можно рассматривать как виртуальные измерения. Таким образом, ребро между двумя узлами является распределением вероятности по относительным местоположениям двух позиций, обусловленным их взаимными измерениями. В общем случае модель наблюдения является мультимодальной и, следовательно, гауссовское предположение не выполняется. Это означает, что одно наблюдение может привести к тому, что несколько краев потенциала свяжут разные позы в графе, а связь с графом сама по себе будет описываться как распределение вероятности. Непосредственное рассмотрение этой мультимодальности в процессе оценки привело бы к комбинаторной сложности. В результате этого большинство практических подходов ограничивают оценку наиболее вероятной топологией.

На рисунке 5 показано как получена и оптимизирована карта с помощью Graph-Based Slam.

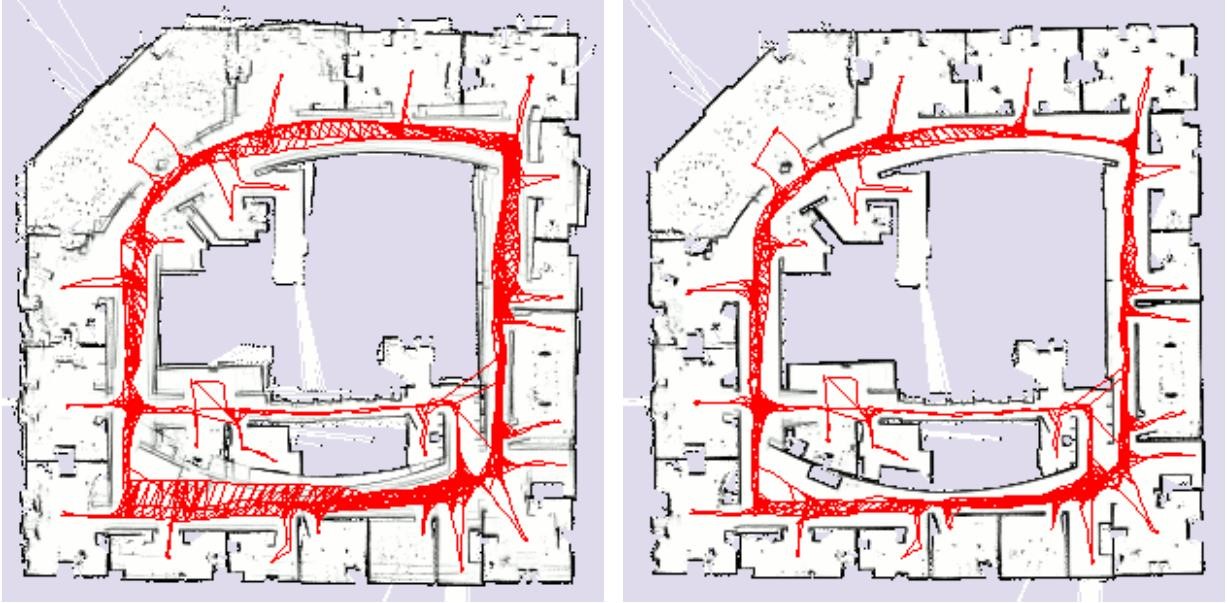


Рисунок 5 – Оптимизация карты с помощью Graph-Based SLAM

Несмотря на то что в данной среде, после тестов оказалось 1802 узла, и 3546 ребер, оптимизация происходит за 100мс на стандартном ноутбуке, что даже позволяет ее производить на каждом шаге робота, а не после обнаружения замыкания цикла.

## 2.3 Gamma-SLAM

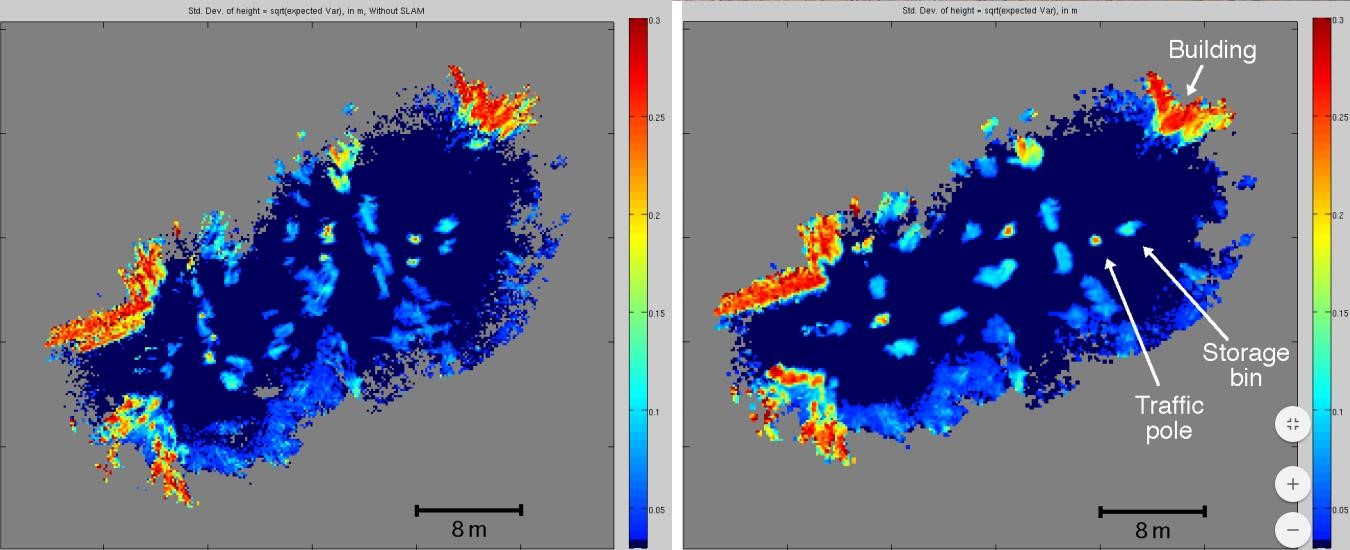
Gamma-SLAM используется для неструктурированных открытых пространств [10]. Также используется другая техника построения карт хранящая апостериорное распределение высоты в каждой ячейке карты, в отличии от других алгоритмов. С помощью Rao-Blackwellized particle filter [8] вычисляется распределение положения робота, при этом каждой позиции соответствует своя карта.

Алгоритм тестировался на модели робота LAGR hHerminator [11]. На него установлен акселерометр и две стереопары, 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.

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



#### Рисунок 6 – Полигон испытаний робота



#### Рисунок 7 – Полученная карта высот

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

## 2.4 Иные реализации SLAM

Стоит отметить ряд реализаций, использующих лазерные дальномеры. В работах строятся весьма точные 2D-карты больших закрытых пространств. Как было отмечено, датчики, применяемые в этих работах, весьма дорогие. Точность карт высока, время работы алгоритмов неизвестно. Карта, полученная алгоритмом DP-SLAM [12] изображена на рисунке 8.

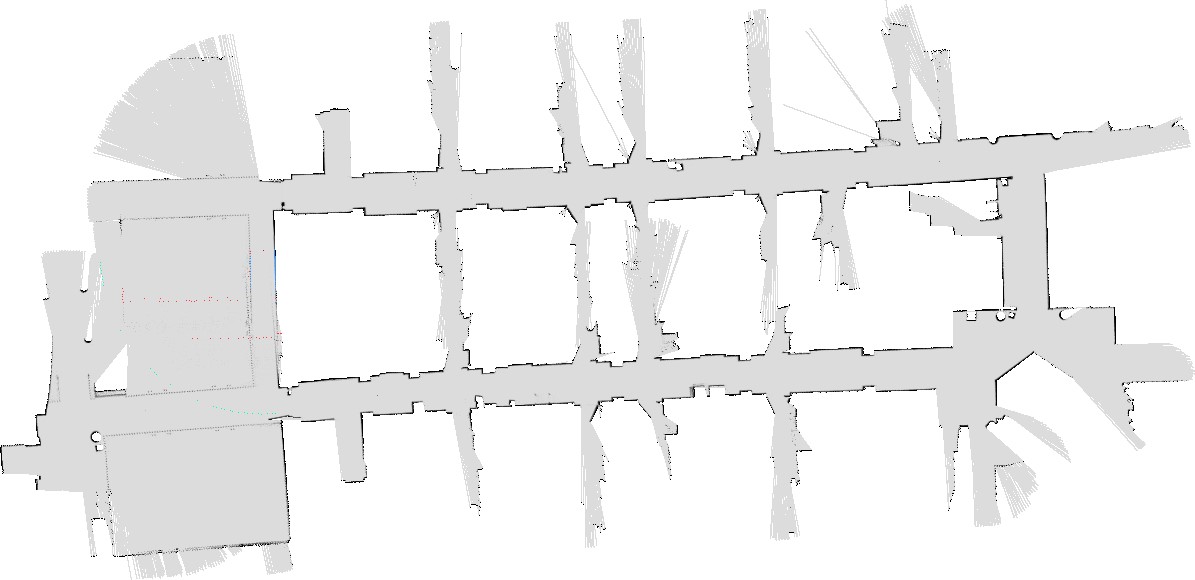
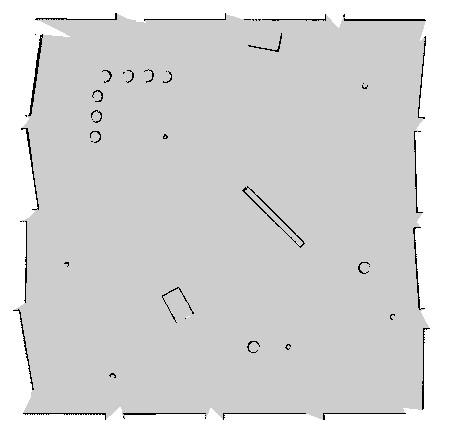


Рисунок 8 – Карта, полученная с помощью DP-SLAM

На рисунке 9 показана работы библиотеки Hector SLAM операционной системы ROS [13], которая предназначена для робототехники.



#### Рисунок 9 – Карта, полученная в ROS

На вход подавались данные с LIDAR, в результате работы данной библиотеки все построения происходят в реальном времени. Следует отметить что работает без данных одометрии. В данной операционной системе существуют различного вида библиотеки для управления серво моторами и другой электроники. Чаще всего данная операционная система устанавливается на микрокомпьютеры Raspberry pi.

# 3 Реализованный метод

В работе было реализовано определение местоположения робота и построение карты, а также различные стратегии поведения робота при перемещении в окружающей среде.

## 3.1 Постановка задачи

Требуется разработать программу в среде Visual Studio 2017, на языке C#, в который должна быть возможность моделировать и визуализировать поведение робота в закрытой среде. Необходимо чтобы робот был оснащен LIDAR, который обладает углом охвата 360 градусов, а также должна быть возможность изменять его угловое разрешение. Должно быть реализовано движение робота вперед и назад, относительно его передней части, также робот должен поворачиваться на любой градус. Робот должен иметь форму квадрата. Должны быть реализовано автономное движение робота, с наименьшим количеством возвратов в те позиции, в которых он был, робот должен объезжать препятствия. В то время как робот перемещается с одной позиции на другую, с помощью данных полученных от LIDAR должна строиться карта местности.

## 3.2 Реализация

### 3.2.1 Перемещение робота

Для того, чтобы робот совершил поворот необходимо повернуть его точки относительно центральной. Так как робот имеет форму квадрата, то повернуть следует только точки, расположенные на углах. Для этого был реализован класс Mechanic который содержит в себе ключевые точки робота, и точки, обозначающие переднюю часть робота, то есть в каком направлении робот должен двигаться. Чтобы робот мог перемещаться по среде был реализован метод, который позволяет роботу двигаться вперед, робот не может двигаться вправо, он может повернуться на 90 градусов, например, и проехать прямо. Для того чтобы робот мог таким образом двигаться необходимо найти нормированный вектор и сместить робота вперед. Реализация нахождения нормированного вектора [13]:

static public PointF nvec (PointF dot1, PointF dot2) { float x = dot1.X - dot2.X; float y = dot1.Y - dot2.Y;

float length = (float)Math.Sqrt(x \* x + y \* y); PointF tmp = new PointF(x / length, y / length); return tmp;}

Где dot1, dot2 точки между которыми необходимо найти

нормированный вектор. Чтобы найти нормированный вектор необходимо найти расстояние между точками и поделить вектор на расстояние.

Таким образом прибавляя данный вектор к каждой точке текущего положения робота робот будет двигаться вперед относительно передней части.

Реализация поворота точки относительно другой на указанное количество градусов [13]:

static public PointF Turndot (PointF dot, PointF centr, double angle){ angle = DegreeToRadian(angle); float xt = dot.X;

dot.X = centr.X + (dot.X - centr.X) \* (float)Math.Cos(angle) -

(dot.Y - centr.Y) \* (float)Math.Sin(angle); dot.Y = centr.Y + (dot.Y - centr.Y) \* (float)Math.Cos(angle) + (xt - centr.X) \* (float)Math.Sin(angle);

return dot;}

Где dot точка, которую поворачиваем относительно точки centr на angle градусов. Метод DegreeToRadian переводит градусы в радианы. Для того чтобы повернуть точки пользуемся матрицей поворота в двумерном пространстве. Для того чтобы повернуть робота необходимо повернуть каждую его точку.

### 3.2.2 LIDAR

Для моделирования работы LIDAR был реализован класс Lidar. Данный класс содержит поля описывающие центральную точку и точку, которая имеет нулевой градус и находится на достаточно большом расстоянии от центральной. В работе используется LIDAR, который не имеет ограничений по расстоянию, а также получает на вход модель окружающей среды в формате bmp. Для того чтобы просканировать область строится уравнение прямой между центральной точкой и точкой, находящейся на далеком расстоянии и имеющую нулевой градус относительно робота. В этом уравнение необходимо пройтись по каждой точке и сравнить цвет если он есть значит необходимо вычислить расстояние от полученной точки до центральной и добавить в список расстояний. Затем точку имеющую нулевой градус необходимо повернуть на градус равный угловому разрешению LIDAR, построить уравнение прямой между новой точкой и центральной затем проделать такие же действия, как и с точкой на нулевом градусе. Необходимые действия делать до тех пор, пока не будет отсканирована область на возможное количество градусов в зависимости от углового разрешения LIDAR, таким образом мы получаем массив дистанций в данной точке среды. Разместив LIDAR в центре робота, и передвигая и поворачивая с роботом мы получаем набор дистанций для различных положений робота.

Реализация сканирования области в указанной точке:

public List<int> Scan(int x, int y){

List<int> listdistances = new List<int>(); PointF[] tmp = new PointF[2]; tmp[0].X = x; tmp[0].Y = y; tmp[1].X = x;

tmp[1].Y = -20000000; for (int i = 0; i < 360 / Resolution; i++)

{ tmp[1].X = x; tmp[1].Y = -20000000;

tmp[1] = Geometry.Turndot(tmp[1], new PointF(x,y), i \*

Resolution); double time = Math.Max(Math.Abs(tmp[0].X - tmp[1].X),

Math.Abs(tmp[0].Y - tmp[1].Y));

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

{ double delta = j / time; double a = delta \* (tmp[1].X - tmp[0].X) + tmp[0].X; double b = delta \* (tmp[1].Y - tmp[0].Y) + tmp[0].Y; Color cl = bmp.GetPixel(Convert.ToInt32(a),

Convert.ToInt32(b));

if (cl.A == 255){

listdistances.Add(Geometry.Distance(new

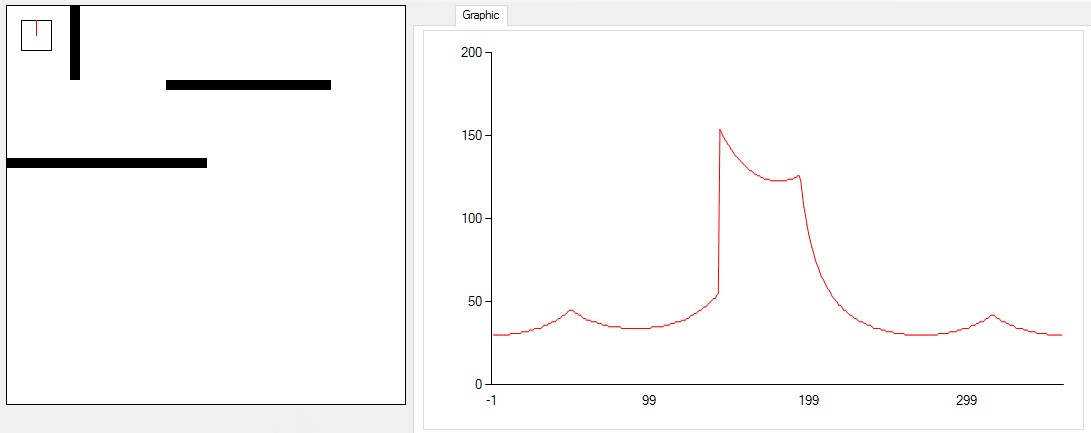
PointF(tmp[0].X,tmp[0].Y), new

PointF(Convert.ToInt32(a), Convert.ToInt32(b))));

break; }}} return listdistances;}

Где x, y координаты в которых производится сканирование.

Пример работы LIDAR показан на рисунке 10.



#### Рисунок 10 – Пример работы LIDAR

На данном рисунке показан график который изображает расстояние до препятствий, сам LIDAR находится в центре робота.

### 3.2.3 Моделирование работы робота

Был реализован класс Robot который содержит объекты классов Mechanic и Lidar. Данный класс содержит класс Drawing, который занимается отрисовкой робота. В данном классе можно визуализировать и моделировать движение робота, его поведение в рабочей среде отрисовка карты. Чтобы измерить на сколько сместился робот относительно изначальной позиции, был заведен стек операций робота. Если робот повернулся относительно предыдущего состояния, то необходимо вычислить на сколько повернулся робот. Чтобы вычислить на сколько градусов повернул робот нужно сравнить индексы максимальных индексов, полученных с помощью LIDAR, на текущем и предыдущем положении, тогда разница этих индексов позволит вычислить на сколько повернул робот. Реализация метода, который позволяет узнать на сколько повернул робот:

static public int Whatangle(List<int> arr1, List<int> arr2){ int i = Max(arr1); int j = Max(arr2);

if (i - j >= arr1.Count()/2) return (i - j) - arr1.Count(); else if ((i - j) <= -(arr1.Count() / 2)) return arr1.Count() + (i - j); else return i - j;}

Где arr1 – список дистанций, полученный на предыдущем шаге, arr2 – список дистанций в текущем состоянии, Max(List<int>) – метод возвращающий индекс максимального значения в списке. Если полученная разница будет больше 180 градусов, то есть смысл переделать ее в отрицательное значение, таким образом мы выявляем повернулся робот вправо или влево.

Если робот проехал, а не повернул, то необходимо узнать на сколько пикселей он проехал, для этого необходимо сравнить значения 0 градуса LIDAR на предыдущем шаге и на текущем, таким образом разница дистанций вернет информацию о том насколько сместился робот. Реализация метода, который позволяет узнать сколько робот проехал:

static public List<int> whatmove(List<int> arr1, List<int> arr2){ List<int> wm = new List<int>(); wm.Add(arr1[arr1.Count() / 2] - arr2[arr2.Count()/2]); wm.Add(arr1[0] - arr2[0]); return wm;}

Где arr1 – список дистанций на предыдущем шаге, arr2 – список дистанций на текущем шаге.

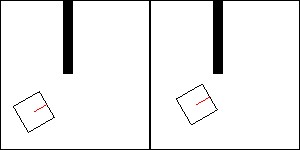
Метод моделирующий работу робота, и записывающий в стек его действия:

public void Move(int step){ List<int> tmp0 = Scan(); md.HideRobot(); meh.StepMehanicsLidar(step); md.DrawRobot(); List<int> tmp1 = Scan(); ma.Add("m\n" + Geometry.whatmove(tmp0, tmp1)[0].ToString() +

"\n" + Geometry.whatmove(tmp0, tmp1)[1].ToString()); map.DrawMap1(this.drm(map.distancetodots(tmp1))); md.DrawChart(Scan());}

Где step – размер шага на сколько должен сместиться робот, так как работа робота моделируется, мы можем сразу узнать на сколько он сместился, игнорируя ошибки, возникшие с помощью одометрических вычислений, но все же стремясь к реальным условиям, вычисляется изменения положения робота все равно с помощью LIDAR, ma – список строк содержащий в себе, на сколько робот проехал, также метод DrawMap который позволяет рисовать карту, tmp0 и tmp1 списки с предыдущим положением и текущем соответственно, передаются в метод whatmove для определения на сколько сместился робот.

Пример движения робота показан на рисунке 11.



#### Рисунок 11 – Движение робота

Метод моделирующий поворот робот относительно его центра: public void Turn(int angle){ List<int> tmp0 = Scan(); md.HideRobot(); meh.TurnMehanicsLidar(angle); md.DrawRobot(); List<int> tmp1 = Scan();

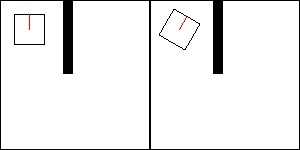
ma.Add("t\n" + Geometry.Whatangle(tmp0, tmp1).ToString());

md.DrawChart(Scan());}

Где angle – величина в градусах, на которую должен повернуть робот.

Методы HideRobot и DrawRobot необходимы для отрисовки робота на Picturebox. Метод Whatangle возвращает то на сколько повернулся угол основываясь на списках сканирования окружающей среды. Метод DrawChart позволяет отображать полученные дистанции в виде графика.

Пример поворота робота показан на рисунке 12.



#### Рисунок 12 – Поворот робота

На данном рисунке показано как робот повернулся на 25 градусов.

### 3.2.4 Построение карты

Класс Map позволяет преобразовывать массив дистанций, полученный с помощью LIDAR в координаты для отображения на PictureBox и реализован метод для построения карты.

Преобразование массива дистанций в точки для отображения:

public List<PointF> distancetodots(List<int> listpoint)

{

List<PointF> tmpdots = new List<PointF>(); int k = 360 / listpoint.Count();

PointF centr = new PointF(this.width / 2, this.height / 2);

for (int i = 0; i < listpoint.Count; i++)

{

PointF mapdot = new PointF(centr.X, centr.Y + listpoint[i]); mapdot = Geometry.Turndot(mapdot, centr, 180 + i\*k); tmpdots.Add(mapdot);

} return tmpdots;

}

Где listpoint полученный список дистанций. Так как мы знаем длин до препятствия то мы берем какую-то точку и говорим, что она является центром, затем берем первый элемент списка и прибавляем к координате x взятой точки длину до препятствия и поворачиваем на 180+i градусов, где i – индекс количества точек, затем проделываем ту же операцию со следующими элементами списка, таким образом мы получаем новый список, в котором содержится координаты.

Пример получаемой карты на разных этапах движения робота изображен на рисунке 13.

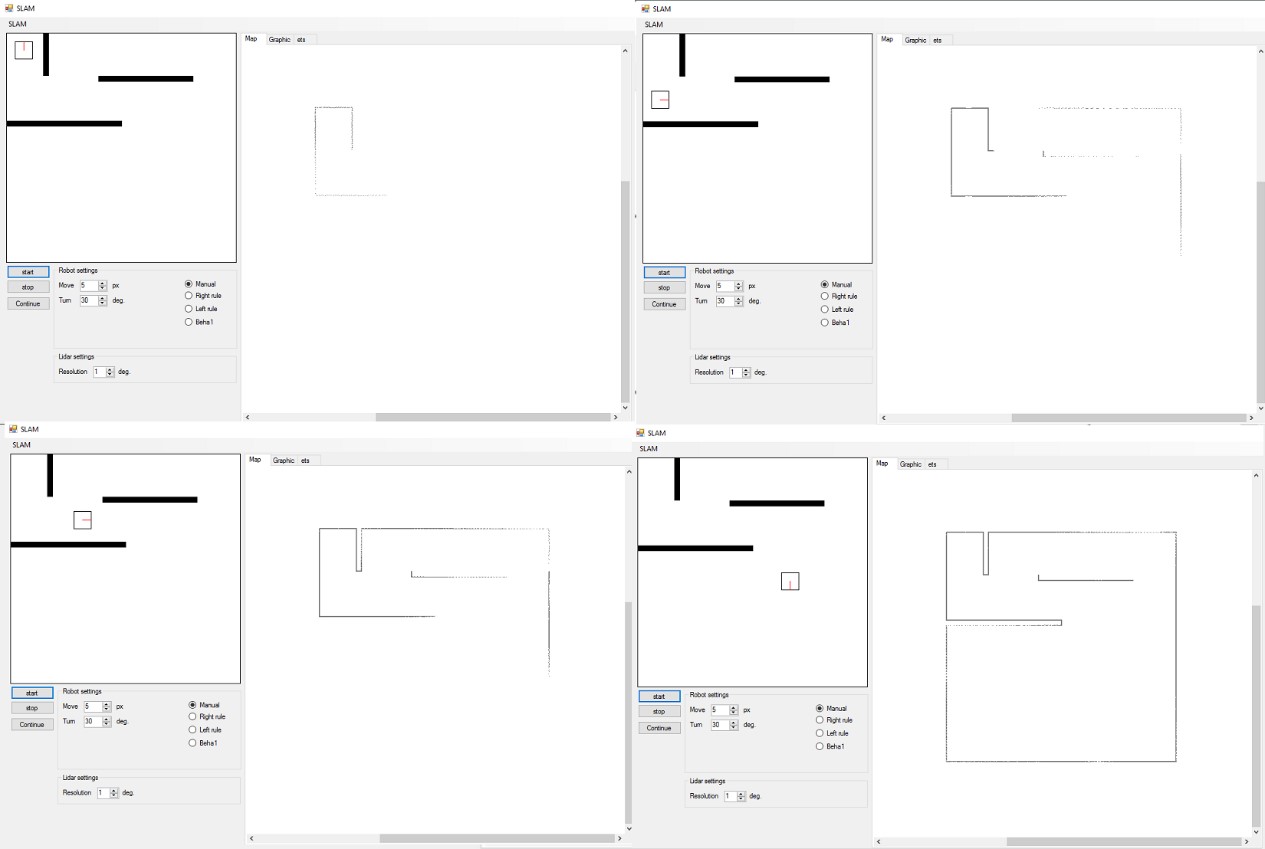


Рисунок 13 – Получаемая в программе карта

Видно, что при помещении робота по окружающей среде происходит построение карты, при этом карта совпадает с окружающей средой.

### 3.2.5 Автономное движение робота

Для того чтобы работ действовал без вмешательства человека, то есть был автономным, необходимо организовать его поведение в окружающей среде. Это является одной из самых сложных задач в данной работе, так как рекомендуется, чтобы робот построил карту всей области, в тоже время не возвращаясь назад, чтобы не было много избыточной информация. Избежание таких возвратов ускорит как работу программы, так и расход ресурсов если применять эти методы для реального робота. Также робот должен объезжать препятствия, так как какое-либо столкновение не особо приветствуется, так как это может создать критическую ситуацию. Существуют способы, когда у робота присутствует датчик касания и робот объезжает таким образом препятствия, но этот тот случай, когда робот не оснащен LIDAR. Для того чтобы предотвратить столкновение с

препятствием необходимо найти максимальное расстояние, на которое может подъехать робот. Для предотвращения этого воспользуемся LIDAR, и если просмотреть расстояния на 0, 45, -45 градусах относительно передней части робота, то в случае если на следующем шаге робот будет на слишком близком расстоянии, то движение вперед невозможно для робота, и ему необходимо либо поворачивать, либо возвращаться назад.

Реализация метода, который позволяет вычислять расстояние до препятствий:

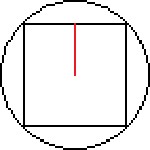
static public int CanMove(List<int> arr, int w){

Boolean f1 = true; Boolean f2 = true; for(int i = 0; i < arr.Count()/8; i++){ if(arr[i] < w){ f1 = false;

break; }} for (int i = (arr.Count()/8)\*7; i < arr.Count(); i++){ if (arr[i] < w){ f1 = false; break; }} for (int i = (arr.Count() / 8) \* 3; i < (arr.Count() / 8) \* 5; i++){ if (arr[i] < w){ f2 = false; break;}} if (!f1 && !f2) return 3; if (f1 && f2) return 0; if (!f1 && f2) return 2; if (f1 && !f2) return 1; return 0;}

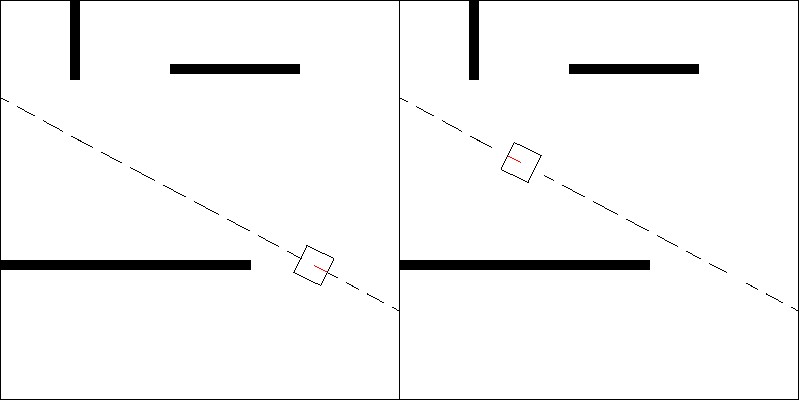
Где arr – список дистанций, полученный с помощью LIDAR, w – расстояние на которое хочет продвинуться робот. В данном методе рассматривается возможность движения робота как вперед, так и назад, в случае если робот может двигаться вперед, метод вернет значение 0, если не может никуда вернет 3, если может вперед, но не может назад вернет 1 и если может назад но не может вперед вернет 2.

В соответствии с рисунком 14 робот имеет квадратную форму, то что выделено кругом вокруг него, это та область, в которую не должны попадать препятствия, так как в случае разворота, робот может столкнуться с ними.



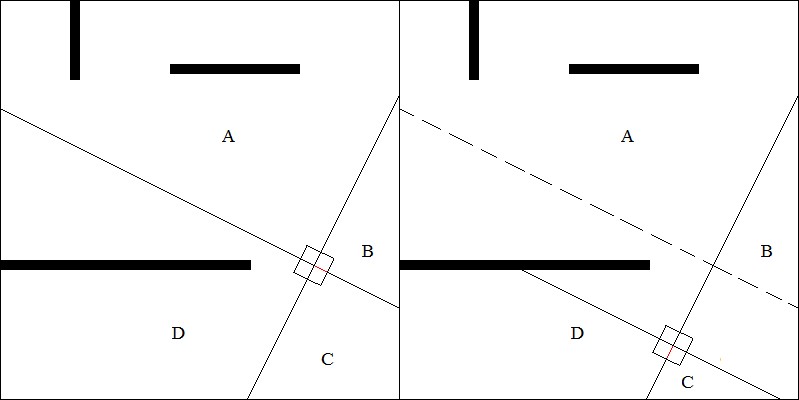
#### Рисунок 14 – Область робота

Также робот должен выбирать себе маршрут для передвижения, что требует достаточно много тестов, так как оптимальный вариант его движения, в зависимости от окружающей среды те или иные способы могут быть не совсем корректными. В данной работе рассматривается следующая модель, находясь в начальной позиции с мощью данных полученных от LIDAR, вычисляется угол, на котором находится самое большое расстояние, робот поворачивается на этот угол и начинается двигаться в этом направлении объезжая препятствия. Для того чтобы вычислить расстояние, которое должен преодолеть робот на этом шаге, можно вычислять различными способами, например, робот может преодолеть 50 процентов наибольшей дистанции, или любое другое число. В данной работе это расстояние вычисляется с помощью случайных чисел, максимальное расстояние делится на случайное число в пределах от 0.1 до 2.0, так как тесты показали, что преодолевать расстояние меньше половины не лучшее решение, а преодолеть расстояние больше выглядит более эффективнее, но возможно, что при на различных средах результаты могут быть совсем другими. Когда робот проехал необходимое расстояние, вновь вычисляется угол с максимальной дистанцией, но тут необходимы дополнительные условия, так как робот может оказаться на самой большой диагонали, и в случае поворота на 180 градусов, робот будет двигаться только по ней, что возможно не позволит найти другие участки карты. Рассмотрим рисунок 15.



#### Рисунок 15 – Диагональ движения

Из рисунка видно что, найдя такую диагональ робот будет двигаться только по ней, но если в процессе движения, например, разделить область перемещения робота на 4 части, и рассматривать максимальные расстояния не на всей области, а на промежутках от 0 до 90, от 90 до 180, от 160 до 270 и от 270 до 360 градусов, и искать там максимальные дистанции, то поведение робота будет гораздо эффективнее, и робот сможет попасть в различные участки. Рассмотрим рисунок 16.



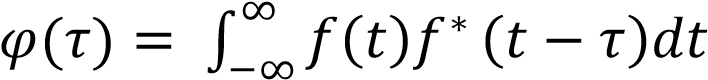
#### Рисунок 16 – Вариация поведения робота

Как мы видим, что если найти наибольшее расстояние не из всей рассматриваемой области, то есть областях A, B, C, D, а рассмотреть каждую область отдельно, и по случаю не возвращать робот назад, так как это дает избыточную лишнюю информацию, то это еще и дает возможность исследовать участки окружающей среды, в которых до этого робот не был.

Таким образом можно подбирать различные стратегии для поведения робота, что позволяет уменьшить количество ненужных действий робота, то есть тех, которое не дают полезной информации, возможно более детально изучить окружающую среду, и разнообразит работу роботу.

### 3.2.6 Самолокализация

Для того чтобы можно было локализовать робота, то есть узнать координаты его положения в среде, необходимо заранее подготовить окружающую среду. С помощью LIDAR необходимо просканировать все окружающую среду, к полученному массиву дистанций, нужно применить автокорреляционную функцию. Автокорреляционная функция – зависимость взаимосвязи между функцией и ее сдвинутой копией от величины временного сдвига. Для детерминированных сигналов автокорреляционная функция сигнала  определяется интегралом

, (14)

где  – величина на которую смещена функция относительна самой себя.

Другими словами, после применения автокорреляционной функции к списку дистанций, полученных от LIDAR, вне зависимости от того на сколько повернут робот в окружающей среде, показания будут одинаковыми, таким образом это облегчает его поиск в пространстве.

Реализация метода позволяющий преобразовать полученный список дистанций к автокорреляционной функции:

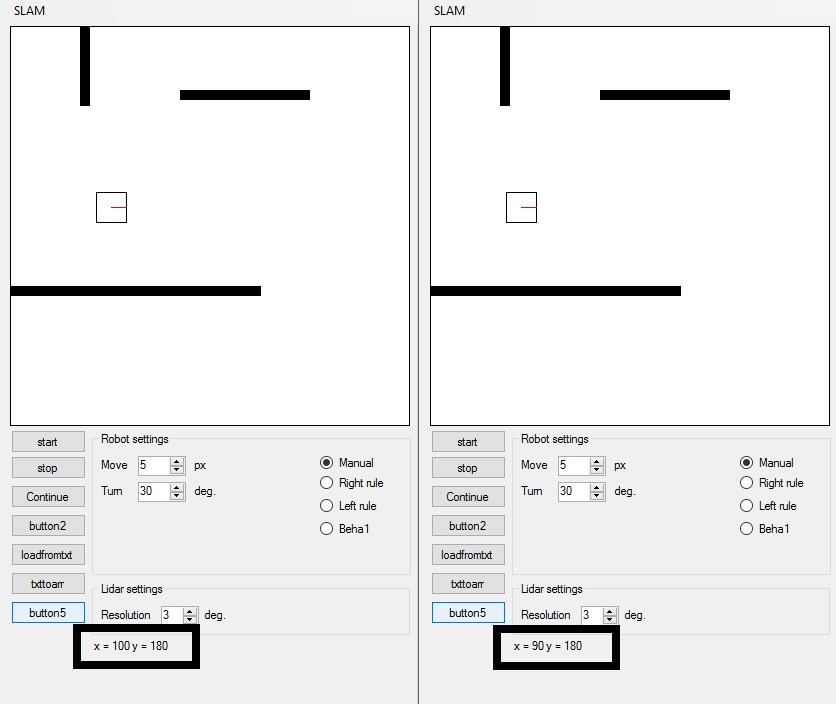
static public List<int> acf(List<int> arr){ List<int> arrtmp = new List<int>(arr); List<int> tmp = new List<int>(); int s = 0; for (int i = 0; i <= arrtmp.Count(); i++){ for (int j = 0; j < arr.Count(); j++){ s += arr[j] \* arrtmp[j];} tmp.Add(s); s = 0; arrayShiftRight(arrtmp); } return tmp;}

Где arr – набор точек к которому необходимо применить фильтр, для того чтобы получить автокорреляционную функцию необходимо найти сумму произведения сначала не сдвинутых, а затем сдвигать массив на один элемент циклически.

При сканировании всей среды не обязательно хранить список дистанций для каждой позиции. Достаточно вычислить максимальную дистанцию, минимальную дистанцию, и разность индексов максимальной и минимальной длинны. Так как процесс сканирования всей области долгий процесс, то в случае неизменности среды, данные записываются в текстовые файлы, так как для последующего запуска это ускорит работу. Так как анализ каждой точки долгий процесс и занимает большое количество памяти, то сканируется не каждый пиксель, а через 1, то есть, если среда размером 400x400, то необходимо произвести 160000 вычислений, а также поиск в этом текущей позиции, поэтому в случае среды 400x400 если сканировать через один пиксель, то необходимо произвести 40000 вычислений, что конечно влияет на точность обнаружения позиции робота, но это оправдывает, время обработки, и количество хранимых данных.

Для того, чтобы уменьшить погрешность при измерении положения робота, можно применять различные методы SLAM, что позволит более точно координировать робота, но при этом затраты увеличатся. В рамках данной работы различные метода не были применены. Так как среда в которой это использовалось, не требовала этого, и данные о местоположении робота, были вычислены с допустимой ошибкой, но в случае применения данного метода на практике следует использовать это, так как в условиях реальной среды, появляются различные негативные факторы, которые будут приводить к более большим ошибкам, что не позволит достаточно точно измерить позиции робота в данный момент его времени, а также с накоплением ошибки в различные этапы, будут приводить к более грубой ошибке.

Пример определения местоположения показан на рисунке 17.



#### Рисунок 17 – Пример самолокализации

На рисунке выделены координаты робота, и их изменение в зависимости от того как он двигается.

## 3.3 Результаты работы программы

Было проведено два испытания, результаты которых были удовлетворяющими заданию. Первый результат показан на рисунке 18 и рисунке 19, где был взят робот размером 30x30, LIDAR имеет угловое разрешение 3 градуса, в качестве окружающей среды выбрана замкнутая статическая среда, с тремя препятствиями размером 400x400. На рисунках показаны промежуточные состояния позиций робота.

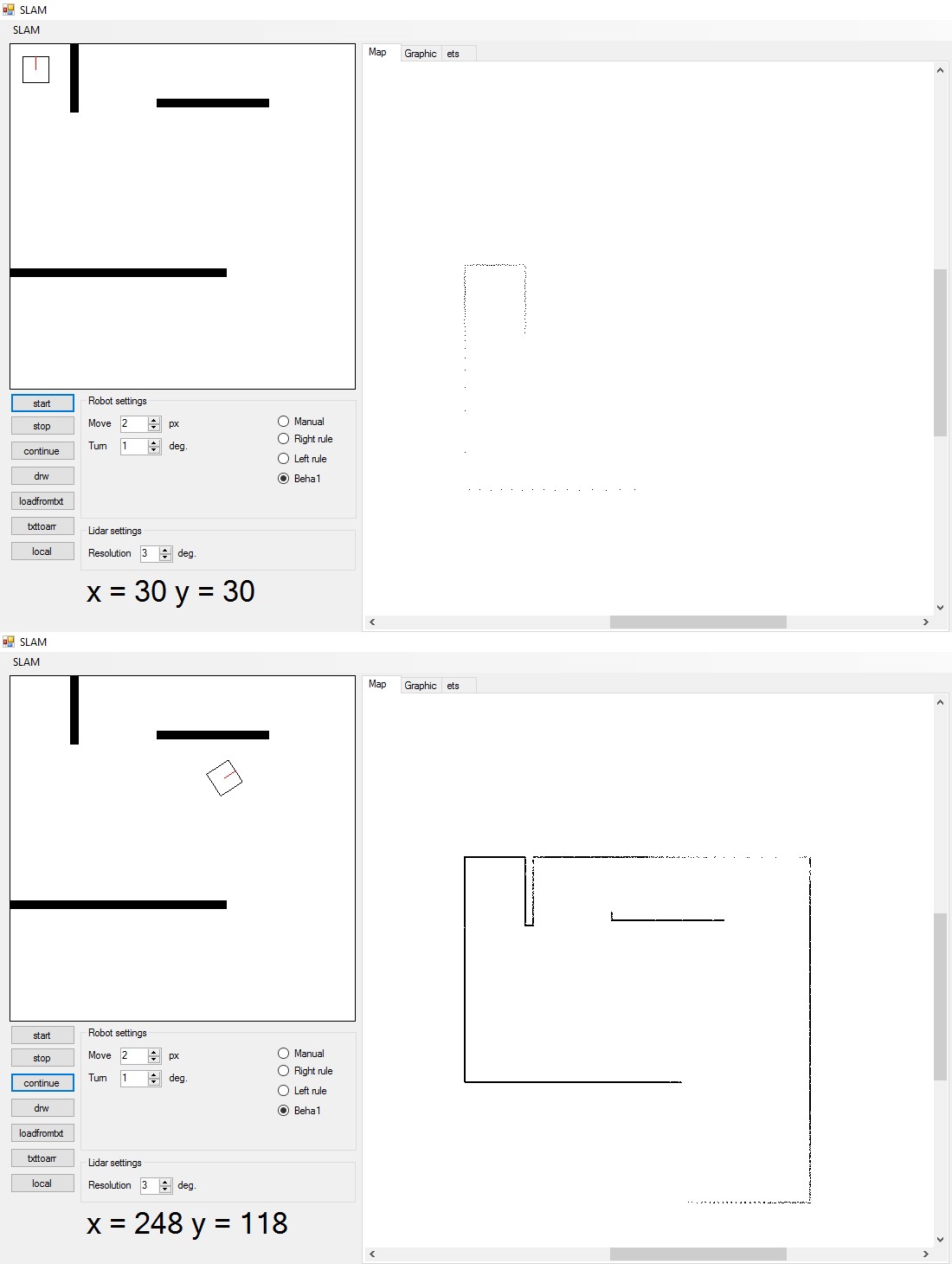
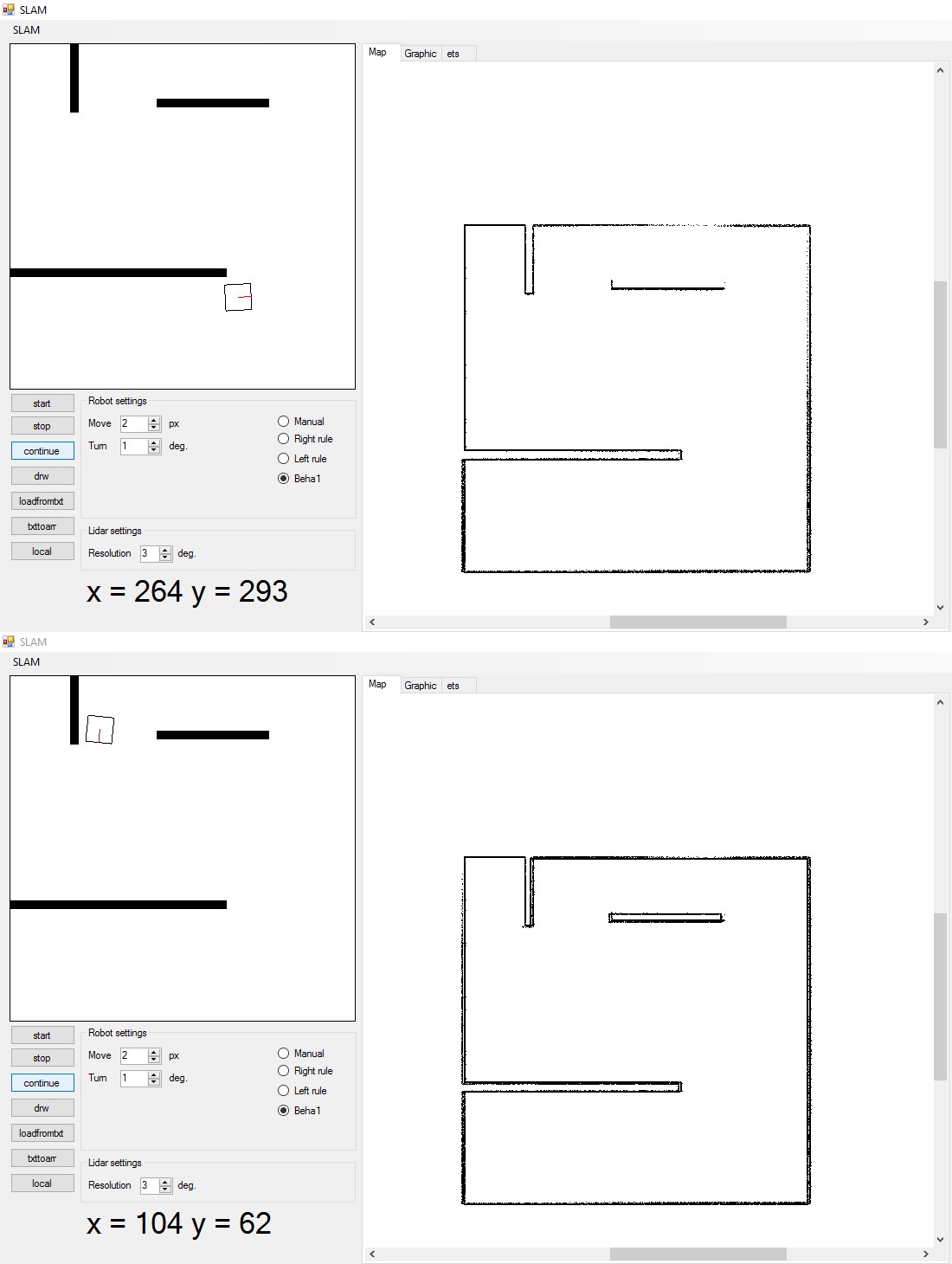


Рисунок 18



#### Рисунок 19

На рисунке 20 и рисунке 21 окружающая среда была изменена, робот в начальной позиции повернут, на карте видно, что поворот робота влияет на получаемую карту.

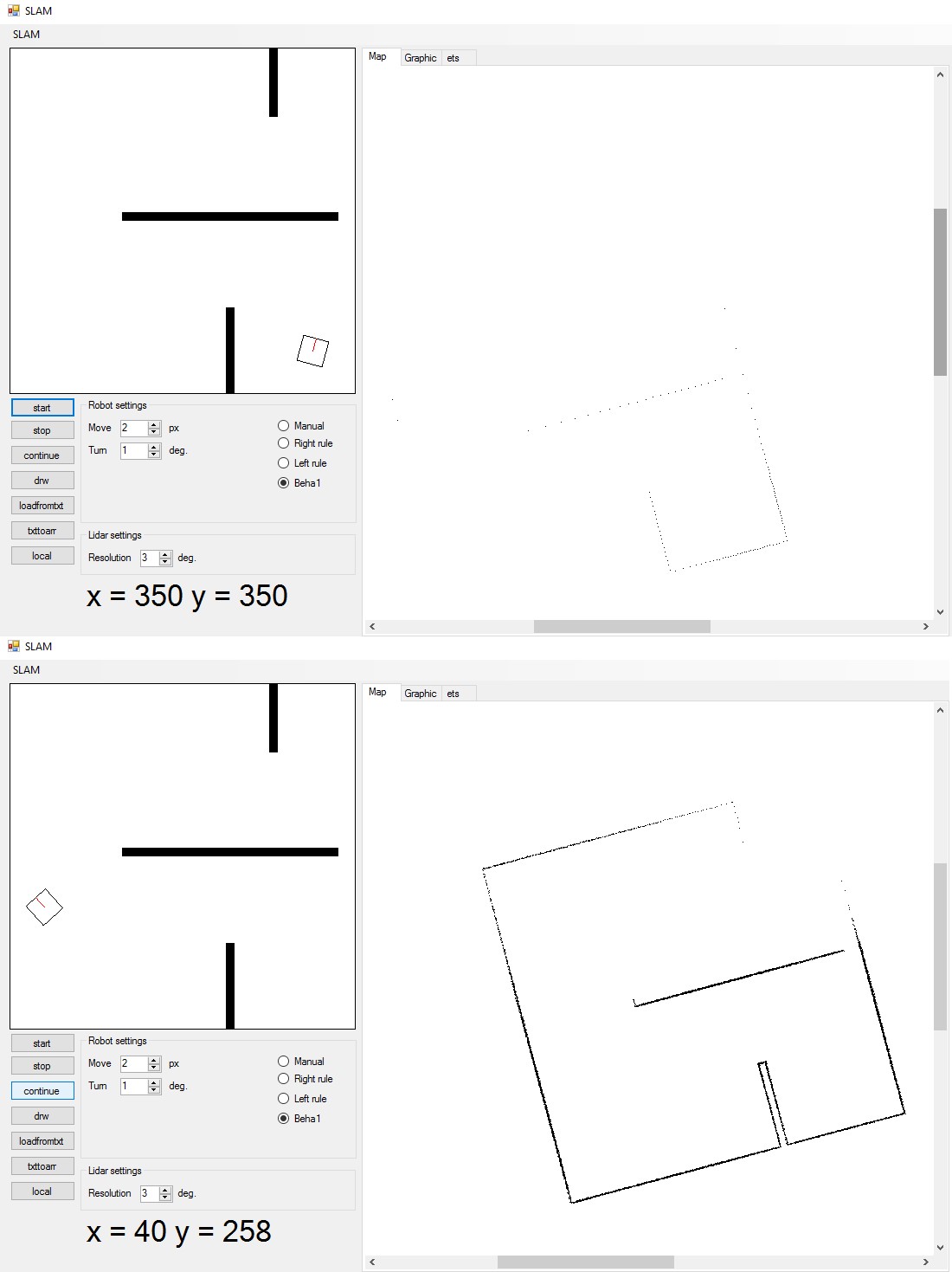
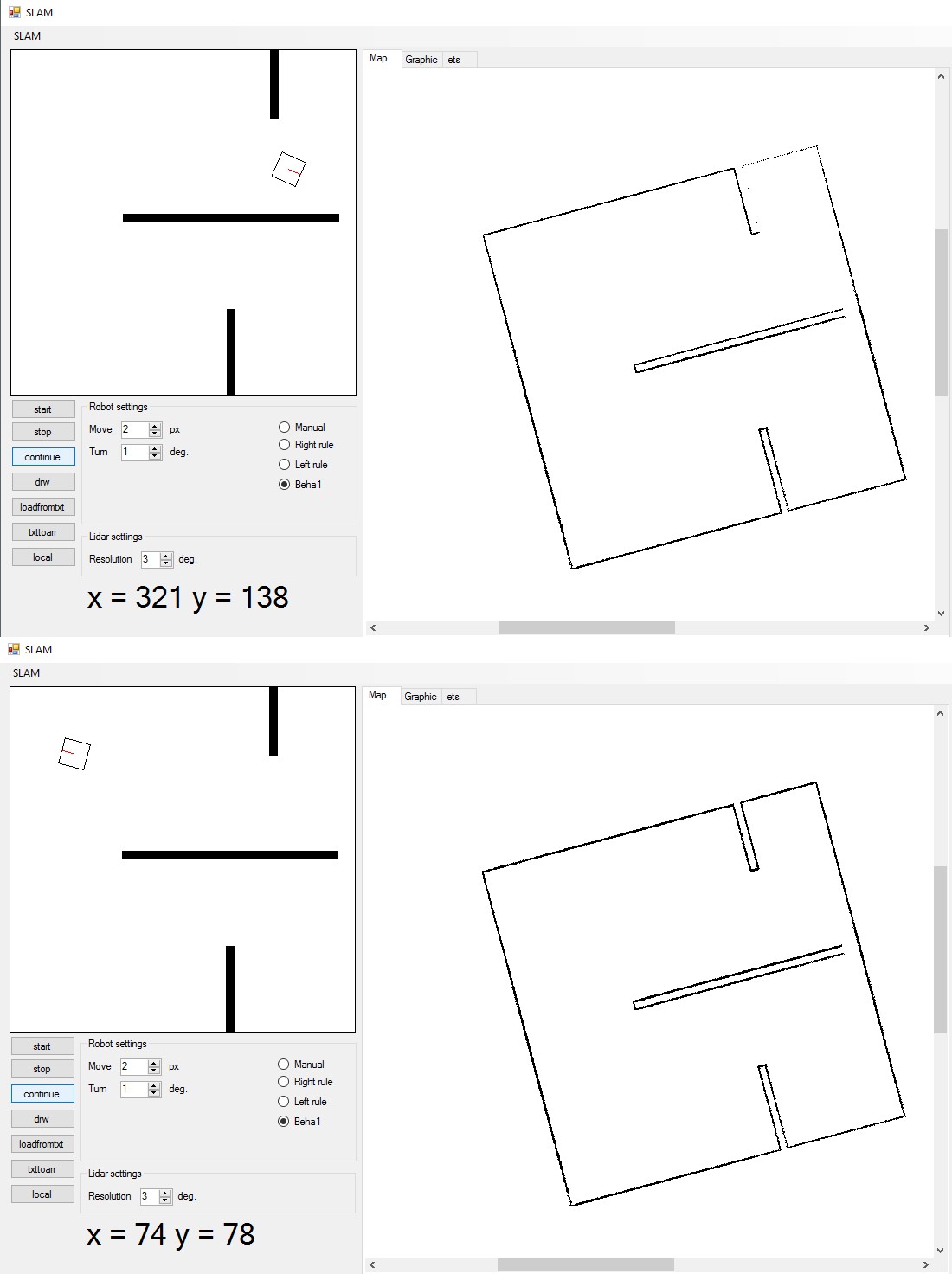


Рисунок 20



#### Рисунок 21

Видно, что роботу удалось побывать во всех местах на карте. Была построена карта местности, определены координаты в каждой позиции, и робот не столкнулся с препятствиями. Следует отметить, что выбранная среда не симметричная, то есть не содержит одинаковых частей на карте, что не создает проблем с локализацией, в случае если окружающая среда симметричная, то необходимо вводить дополнительные условия, но это может не помочь, если ошибка будет допущена при стартовой позиции робота. Также если изменить угловое разрешение LIDAR на 1 градус, то можно добиться более качественной карты, и с меньшим количеством перспективных ошибок, и более точное определение робота в пространстве, но это замедляет работу, и требует больше ресурсов.

# ЗАКЛЮЧЕНИЕ

В процессе выполнения работы, были изучены алгоритмы синхронной локализации и построения карты на основании данных, предоставляемых датчиками робота.

В результате была создана программа, предназначенная для моделирования и визуализации поведения робота, оснащенного LIDAR, в закрытом пространстве. С использованием этой программы можно моделировать автономное движение робота в статической среде, изменяя параметры датчика и задавая особенности окружающей среды. Данная программа позволяет строить карту по информации полученной от датчика и локализовать робота.

Необходимость разработки подобных программ состоит в том, что они позволяют производить испытания на стадии проектирования робота с целью выбора датчиков для его оснащения и с учетом особенностей среды в которой он будет работать.

В дальнейшем представленная программа может быть модифицирована на случай использования различных датчиков и фильтров для построения карты, а также для рассмотрения различных стратегий поведения робота и добавления динамических объектов.

Автономные роботы позволяют исследовать локации, в которые не может попасть человек, например, осмотр шахты после обрушения. Другой пример использования – сфера услуг, например, робот, который перевозит товары на складе.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Riisgaard S., Blas M. R. SLAM for Dummies. [Электронный ресурс].

URL: [http://ocw.mit.edu/NR/rdonlyres/Aeronautics-and-Astronautics/16-](http://ocw.mit.edu/NR/rdonlyres/Aeronautics-and-Astronautics/16-412JSpring-2005/9D8DB59F-24EC-4B75-BA7A-F0916BAB2440/0/1aslam_blas_repo.pdf)

[412JSpring-2005/9D8DB59F-24EC-4B75-BA7A-](http://ocw.mit.edu/NR/rdonlyres/Aeronautics-and-Astronautics/16-412JSpring-2005/9D8DB59F-24EC-4B75-BA7A-F0916BAB2440/0/1aslam_blas_repo.pdf)

[F0916BAB2440/0/1aslam\_blas\_repo.pdf](http://ocw.mit.edu/NR/rdonlyres/Aeronautics-and-Astronautics/16-412JSpring-2005/9D8DB59F-24EC-4B75-BA7A-F0916BAB2440/0/1aslam_blas_repo.pdf) (Дата обращения 14.02.2017)

1. ГОСТ Р ИСО 8373-2014 Роботы и робототехнические устройства.

Термины и определения; введ. 2016-01-01. М.: Стандартинформ, 2015.

1. Дж. Фрайден. Современные датчики. Справочник. М.: Техносфера, 2005, 592 с.
2. Одометрия. [Электронный ресурс]. URL:

<http://robocraft.ru/blog/technology/736.html>(Дата обращения

25.03.2017).

1. Thrun S., Montemerlo M., Koller D., Wegbreit B. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping

Problem. [Электронный ресурс]. URL: <http://robots.stanford.edu/papers/montemerlo.fastslam-tr.pdf>(Дата обращения 30.03.2017).

1. Welch G., Bishop G. An Introduction to the Kalman Filter.

[Электронный ресурс]. URL:

<http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf>(Дата обращения 28.01.2017)

1. Riberio M. I. Kalman and Extended Kalman Filters: Concept, Derivation and Properties. [Электронный ресурс]. URL:

<http://users.isr.ist.utl.pt/~mir/pub/kalman.pdf>(Дата обращения

5.02.2017)

1. Thomas Schön. (Rao-Blackwellized) Particle Filters and Localization.

[Электронный ресурс]. URL: [http://www2.informatik.unifreiburg.de/~grisetti/teaching/ls-slam/lectures/pdf/ls-slam-06-rbpfloc.pdf](http://www2.informatik.uni-freiburg.de/~grisetti/teaching/ls-slam/lectures/pdf/ls-slam-06-rbpfloc.pdf)

(Дата обращения 11.05.2017)

1. Grisetti G., Kummerle R., Stachniss C., Burgard W. A Tutorial on

Graph-Based SLAM. [Электронный ресурс]. URL:

[http://www2.informatik.unifreiburg.de/~stachnis/pdf/grisetti10titsmag.pdf](http://www2.informatik.uni-freiburg.de/~stachnis/pdf/grisetti10titsmag.pdf) (Дата обращения

18.04.2017)

1. 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. [Электронный ресурс]. URL: <https://cseweb.ucsd.edu/~gary/pubs/slam_icra08-FINAL.pdf>(Дата обращения 17.02.2017)
2. Jackel L.DARPA`s LAGR and UPI programs. [Электронный ресурс].

URL:

[http://www.laas.fr/IFIPWG/Workshops&Meetings/49/workshop/06%20j ackel.pdf](http://www.laas.fr/IFIPWG/Workshops%26Meetings/49/workshop/06%20jackel.pdf) (Дата обращения 07.03.2017)

1. Eliazar A.I., Parr R. DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without Predetermined Landmarks.

[Электронный ресурс]. URL:

<http://people.ee.duke.edu/~lcarin/Lihan4.21.06a.pdf>(Дата обращения

05.03.2017)

1. Quigley M., Gerkey B., Conley K., Faust J., Foote T., Leibs J., Berger E., Wheeler R. ROS: an open-source Robot Operating System.

[Электронный ресурс] URL: http://www.robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf (Дата обращения 13.04.2017)

1. Тыртышников Е.Е. Матричный анализ и линейная алгебра: Курс лекций. М.: ИВМ РАН, 2005, 372 с.