МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра компьютерных технологий и систем

Бекетов Дмитрий Данилович

РАЗРАБОТКА АЛГОРИТМА ОПТИМАЛЬНОГО ПУТИ РОБОТА В ЛАБИРИНТЕ С ЗАРАНЕЕ ИЗВЕСТНОЙ СТРУКТУРОЙ

Дипломная работа

Научный руководитель:
Доцент кафедры КТС,
к.п.н А. А. Францкевич

Допущена к защите
«____» _____ 2025 г.

Зав. кафедрой компьютерных технологий и систем
канд. физ.-мат. наук, профессор В.В. Казаченок

Минск, 2025

ОГЛАВЛЕНИЕ

РЕФЕРАТ

Дипломная работа, 51 страница, 16 источников, 26 иллюстраций, 16 формул.

Ключевые слова: НАВИГАЦИЯ РОБОТА; ЛАБИРИНТ; ЛОКАЛИЗАЦИЯ; КАРТИРОВАНИЕ; КОМПЬЮТЕРНОЕ ЗРЕНИЕ; ОПТИМАЛЬНЫЙ ПУТЬ.

Объектом исследования являются существующие методы и инструменты локализации и навигации робота в пространстве, а также нахождения оптимального пути в лабиринте. Средства симуляции и методы компьютерного зрения.

Предметом исследования являются методы локализации, навигации и поиска оптимального пути для робота в лабиринте, а также средства симуляции и компьютерного зрения.

Целью работы является разработка алгоритма нахождения оптимального пути робота в лабиринте, учитывая вид сверху с камеры, дрона, спутника.

Методами исследования являются исследование литературы, тестирование методов и алгоритмов в среде моделирования и симуляции.

Полученные результаты и их новизна: Система, которая анализирует лабиринт, находит в нем кратчайший путь, используя 3 вида алгоритмов и пускает робота по наиболее оптимальному пути.

Достоверность материалов и результатов дипломной работы: использованные материалы и результаты дипломной работы являются достоверными. Работа выполнена самостоятельно.

Областью возможного практического применения является оптимизация логистических процессов на складах крупных компаний.

_____

(подпись студента)

РЭФЕРАТ

Дыпломная праца, 51 старонак, 16 крыніц, 26 ілюстрацый, 16 формул.

Ключавыя словы: НАВІГАЦЫЯ РОБОТА; ЛАБІРЫНТ; ЛАКАЛІЗАЦЫЯ; КАРТАВАННЕ; КАМП'ЮТЭРНАЕ БАЧЭННЕ; АПТЫМАЛЬНЫ ШЛЯХ.

Аб'ектам даследавання з'яўляюцца існуючыя метады і інструменты лакалізацыі і навігацыі робата ў прасторы, а таксама знаходжання аптымальнага шляху ў лабірынце. Сродкі сімуляцыі і метады камп'ютарнага бачання.

Прадметам даследавання з'яўляюцца метады лакалізацыі, навігацыі і пошуку аптымальнага шляху для працы ў лабірынце, а таксама сродкі мадэлявання і камп'ютэрнага гледжання.

Мэтай даследавання з'яўляецца распрацоўка алгарытму знаходжання аптымальнага шляху робата ў лабірынце з улікам выявы зверху з камеры, дрона, спадарожніка.

Метадамі даследавання з'яўляюцца аналіз літаратуры, тэставанне метадаў і алгарытмаў у асяроддзі мадэлявання і сімуляцыі.

Атрыманыя вынікі і іх навізна: Сістэма, якая аналізуе лабірынт, знаходзіць у ім найкарацейшы шлях, выкарыстоўваючы 3 віды алгарытмаў, і накіроўвае робата па найбольш аптымальным маршруце.

Даставернасць матэрыялаў і вынікаў дыпломнай працы: выкарыстаныя матэрыялы і вынікі дыпломнай Працы з'яўляюцца даставернымі. Праца выканана самастойна.

Вобласцью магчымага практычнага прымянення з'яўляецца аптымізацыя лагістычных працэсаў на складах буйных кампаній.

(подпіс студэнта)

SUMMARY

Diploma work, 51 pages, 16 sources, 26 illustrations, 16 formulas.

Keywords: ROBOT NAVIGATION; MAZE; LOCALIZATION; MAPPING; COMPUT
VISION; OPTIMAL PATH.

The object of the research is the existing methods and tools for robot localization and navigation in space, as well as finding the optimal path in a maze. Simulation tools and computer vision methods.

The subject of the research is methods of localization, navigation and finding the optimal path for a robot in a maze, as well as means of simulation and computer vision.

The purpose of the research is the development of an algorithm for finding the optimal path for a robot in a maze, considering an overhead view from a camera, drone, or satellite.

Methods of research are literature reviews, testing of methods and algorithms in a modeling and simulation environment.

The results of the work and their novelty: a system that analyzes a maze, finds the shortest path using 3 types of algorithms, and guides the robot along the most optimal route.

Authenticity of the materials and results of the diploma work: the materials used and the results of the diploma work are authentic. The work has been put through independently.

Recommendations on the usage. Optimization of logistics processes in warehouses of large companies.

(student's signature)

ВВЕДЕНИЕ

В 21 веке робототехника переживает бурное развитие. В современных условиях, мобильные робототехнические системы находят все более широкое применение в различных сферах человеческой деятельности. В промышленном секторе отмечается устойчивый спрос на многофункциональные роботизированные комплексы, способные выполнять сложные производственные задачи. На потребительском рынке прослеживается активный интерес к бытовым роботам, таким как интеллектуальные системы уборки и интерактивные роботы-компаньоны. Особое значение мобильные робототехнические системы приобретают в области безопасности и спасательных операций. Автономные robotic платформы демонстрируют высокую эффективность при выполнении задач мониторинга и поисковых операций, что обусловлено их способностью к длительной непрерывной работе в различных условиях. Люди создают и развивают роботов, чтобы те:

— Высвобождали человека в процессе производства продукции от тяжелых видов работ, а также его пребывания в экстремальных условиях (загрязненной среде, химической среде, опасной для жизни и т.п.).

— Существенно повышали производительность труда при выполнении операций в процессе производства продукции.

— Значительно повышали качество продукции, производимой в промышленном производстве.

— Снижали себестоимость продукции, производимой на определенном промышленном предприятии.

АКТУАЛЬНОСТЬ ТЕМЫ ИССЛЕДОВАНИЯ

«Подобные устройства в идеале должны уверенно перемещаться в незнакомой и непредсказуемой обстановке реального мира. Пока основной проблемой всех ныне существующих мобильных аппаратов, перемещающихся самостоятельно, без управления со стороны человека, остается навигация. Для успешной навигации в пространстве бортовая система робота должна уметь строить маршрут, управлять параметрами движения (задавать угол поворота колес и скорость их вращения), правильно интерпретировать сведения о окружающем

мире, получаемые от датчиков, и постоянно отслеживать собственные координаты» [4]. Алгоритмы навигации и движения мобильных роботов в неизвестных лабиринтах представляют собой важную область исследований в робототехнике. И мир активно развивается и внедряет разработки в этом направлении.

«Это доказывает опыт Amazon: роботы-кладовщики делают за 15 минут то, на что у людей уходит 60-75 минут. Благодаря этому компания снизила операционные расходы на 13 складах на 20 процентов — это 22 миллиона долларов на каждый склад. Когда роботы появятся на всех 110 складах компании, экономия составит 800 миллионов долларов» [10].

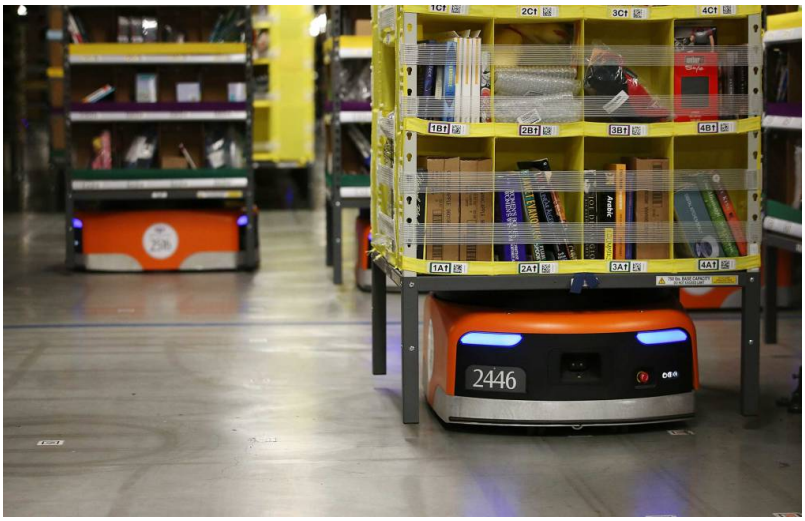


Рисунок 1 - Роботы на складе Amazon

Я считаю важным внедрение подобных технологий на складах нашей страны.

ЦЕЛЬ И ЗАДАЧИ ИССЛЕДОВАНИЯ

Целью данной работы является разработка алгоритма навигации и движения робота в лабиринте с заранее известной структурой. Робот получает изображение лабиринта сверху, после чего в памяти выстраивает оптимальный маршрут передвижения по лабиринту, то есть исключая все тупики. Для осуществления поставленной цели необходимо решить следующие задачи.

1. Провести обзор научной литературы по алгоритмам навигации робота в лабиринтах.

2. Рассмотреть всевозможные методы картирование и локализации пространства.

3. Разработать алгоритм навигации и движения робота в лабиринте.
4. Выбрать среду для тестирования и визуализации алгоритма.
5. Протестировать алгоритм на различных видах лабиринтов.
6. Проанализировать полученные результаты. Выявить возможные недостатки и пути их устранения.

Алгоритмы навигации и движения робота в лабиринте

За долгие годы люди научились представлять многие обычные вещи и явления в виде графов. Благодаря этому достижению мы способны оптимизировать доставку грузов или перелет пассажиров на самолете, используя при этом множество алгоритмов, реализованных на графах. В настоящее время в нашем распоряжении такие эффективные алгоритмы, как: A*, алгоритм Дейкстры и алгоритм Беллмана. И как оказалось, лабиринт также можно представить в виде взвешенного графа и применять все вышеперечисленные алгоритмы на нем. Однако для того чтобы применить данные алгоритмы, необходимо полностью исследовать весь лабиринт и оказаться в начальной точке.

В данной работе будет реализован алгоритм, который будет считывать информацию о лабиринте с камеры. Камера находится над лабиринтом и, используя методы компьютерного зрения и алгоритмы поиска кратчайшего пути, находит оптимальный путь прохождения лабиринта.

Соревнования Micromouse

Micromouse — конкурс для маленьких роботов-мышей, которые должны быстрее всех найти путь в центр лабиринта. Лабиринт не очень большой, его размер 32x32 квадрата (раньше было 16x16) с длиной грани 9 см. Высота стенок каждой ячейки 2,5 см, толщина — 0,6 см. Micromouse должна отслеживать, где она находится, обнаруживать стены во время исследования, составлять карту лабиринта и обнаруживать, что она достигла цели.

Составление карты важно, ведь мышь может несколько раз пройти лабиринт, пока не найдёт оптимальный маршрут от начала до конца. И затем проходит его максимально быстро. Участникам не разрешается обновлять код своих робомышей после того, как демонстрируется схема лабиринта. Вы могли бы подумать, что у роботизированной мыши не так много способов пройти лабиринт, но за свою почти 50-летнюю историю участники соревнований Micromouse неоднократно доказывали ошибочность этого предположения.

Обычные алгоритмы поиска оптимального маршрута используют вариации алгоритма Беллмана, алгоритма Дейкстры, алгоритма поиска A*, различных алгоритмов обхода дерева и обхода графов.

ГЛАВА 1

РАЗРАБОТКА АЛГОРИТМА НАХОЖДЕНИЯ ОПТИМАЛЬНОГО ПУТИ

1.1 Постановка задачи

Для того чтобы определиться с поставленной задачей, рассмотрим, какие входные данные мы имеем. Имеем некоторое пространство, которое будем называть лабиринт. Лабиринт — это сложная структура, которая представляет собой сеть путей или ходов, предназначенных для перемещения внутри неё. Целью может быть достижение конкретной точки, например, выхода или центра лабиринта. В нашем случае лабиринт — это сложная структура, имеющая один вход и один выход. Также имеем изображение лабиринта сверху, получаемое с камеры. И первоначальное положение робота, которое мы не знаем относительно самого лабиринта. Задача состоит в том, чтобы на основе входных данных, которые мы имеем, разработать систему навигации и прохождения робота в лабиринте. Для того чтобы достичь решения поставленной задачи, сформулируем список вопросов, на которые система должна отвечать:

— Где робот?

— Куда роботу нужно прийти?

— Есть ли карта лабиринта?

— Какой самый короткий путь к цели?

— Как роботу избежать столкновений с препятствиями?

1.2 Методы локализации

Первый вопрос, на который будем отвечать, где находится робот? Будем исходить из условий среды, и так мы имеем: лабиринт, статичная камера над лабиринтом и робот в лабиринте. Следовательно, мы имеем статичную кар-

тинку лабиринта, в котором изменяется только местоположение робота. Будем использовать метод обнаружения движущихся объектов с помощью OpenCV с использованием обнаружения контуров и вычитания фона.

Обнаружение движущихся объектов представляет собой важную задачу в области компьютерного зрения, находящую применение в системах видеонаблюдения, мониторинга дорожного движения и навигации автономных роботов. В данной главе рассматриваются методы обнаружения движущихся объектов с использованием библиотеки OpenCV, основанные на комбинации вычитания фона и обнаружения контуров.

Как можем увидеть на практике, эффективное обнаружение движущихся объектов достигается путем последовательного применения методов вычитания фона и обнаружения контуров. Вычитание фона позволяет получить бинарную маску переднего плана, где движущиеся объекты выделяются белыми пикселями на черном фоне. Последующее применение методов обнаружения контуров дает возможность точно определить границы и положение движущихся объектов.

В библиотеке OpenCV реализация данного подхода начинается с инициализации фоновой модели путем захвата эталонного изображения без движущихся объектов. Затем фоновая модель постоянно обновляется для адаптации к изменениям в сцене, таким как переменное освещение или появление новых статических объектов.

Для выделения движущихся объектов используется пороговая обработка с заданным значением чувствительности. После получения маски переднего плана применяются алгоритмы обнаружения контуров, позволяющие идентифицировать и локализовать движущиеся объекты на изображении. Первым шагом обнаружения движущихся объектов является вычитание фона. Использование статической камеры — распространённый и широко используемый метод для создания маски переднего плана (а именно двоичного изображения, содержащего пиксели, принадлежащие движущимся объектам в сцене).

1. Фоновая инициализация.

2. Фоновое обновление.

Рассмотренный подход может быть усовершенствован за счет повышения

устойчивости к изменяющимся условиям освещения и улучшения точности обнаружения при частичном перекрытии объектов. Дальнейшие исследования в этом направлении позволят создать более надежные и эффективные системы обнаружения движущихся объектов для различных практических применений.

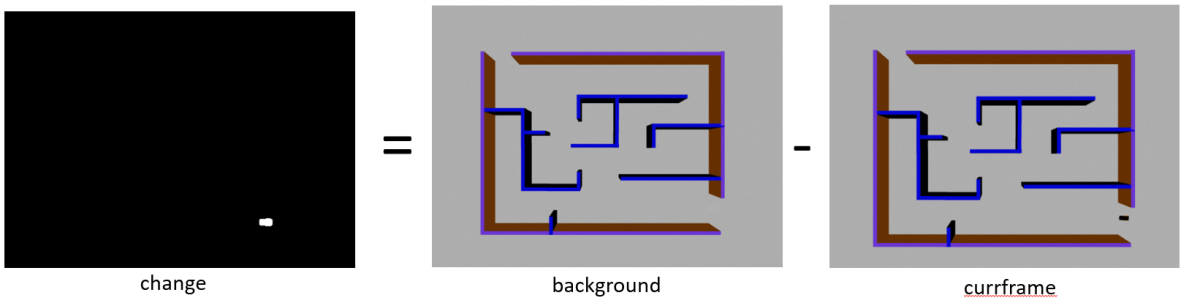


Рисунок 1.1 - Вычитание фона с использованием OpenCV

Как можем увидеть на иллюстрации, имея задний план в виде лабиринта и текущее изображение лабиринта вместе с роботом, воспользуемся попиксельным вычитанием и найдем разницу. И для того чтобы из полученной разницы изображений получить битовую маску, где область робота будет выделена белым, воспользуемся функцией threshold из библиотеки OpenCv2, которая использует пороговое значение. Пороговое значение 15 означает то, что все пиксели с интенсивностью больше 15 станут белыми (255), а остальные — чёрными (0). После чего, на полученной маске, выделим контуры всех объектов и найдем максимальный по площади — это и будет робот.

1.2.1 Координаты робота

Но как получить координаты робота в пространстве? Зная контур робота, можем вычислить центроид области, которую огибает контур. Центроид или центр масс объекта — это точка, которая представляет собой среднее положение всех точек объекта. В контексте изображения, где объект представлен как контур или набор пикселей, центроид можно вычислить как среднее положение всех пикселей в этом контуре. OpenCV2 предоставляет удобный функционал для вычисления центроида с помощью моментов.

Если представить объект, как множество точек, каждую из которых можно рассматривать как маленькую массу, то:

— M_{10} и M_{01} — это моменты, которые описывают положение "массы"объекта относительно осей координат x и y.

$$M_{10} = \sum_x \sum_y x \cdot f(x, y), \tag{1.1}$$

$$M_{01} = \sum_x \sum_y y \cdot f(x, y); \tag{1.2}$$

— M_{00} — это "общая масса"объекта (площадь, если рассматривать объект как 2D-фигуру).

$$M_{00} = \sum_x \sum_y f(x, y) \tag{1.3}$$

Центроид (C_x, C_y) вычисляется как:

$$C_x = \frac{M_{10}}{M_{00}}, \tag{1.4}$$

$$C_y = \frac{M_{01}}{M_{00}}; \tag{1.5}$$

После чего, выполним смещение начала координат и коррекцию выхода за пределы. После всех преобразований получаем центроид робота в системе координат лабиринта.

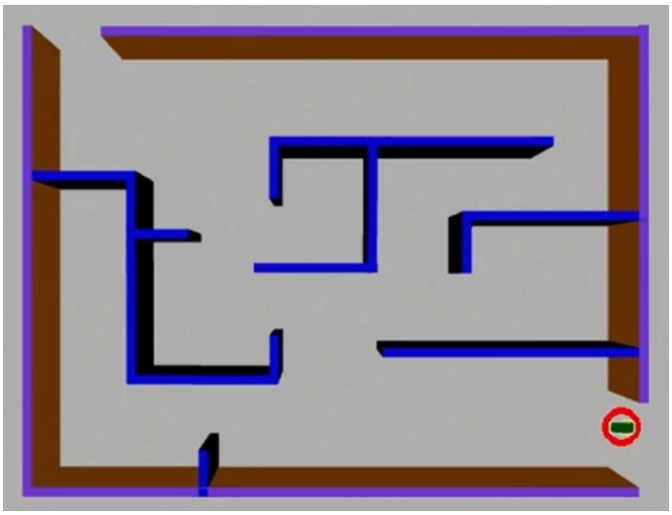


Рисунок 1.2 - Выделение модели робота

1.2.2 Алгоритм детекции краев Джона Ф. Кэнни

«Поиск контуров — термин, обозначающий набор математических методов, направленных на выявление точек в цифровом изображении, в которых яркость изображения резко меняется. Эти точки обычно организованы в виде набора кривых линий и называются краями, границами или контурами. Изменение яркости изображения может соответствовать: различным материалам, различию в освещении разных частей сцены, перепадам глубины или изменению ориентации поверхности. В идеальном случае определение краев помогает установить границы и форму объекта. Выделенные края могут быть двух типов: независящие и зависящие от точки зрения. Независящие границы отображают такие свойства, как цвет и форма поверхности. Зависящие могут меняться в разных точках обзора и отображают геометрию сцены.

В 1986 году Джон Канни разработал детектор границ, оптимальный для трех критериев: низкий уровень ошибок, правильная локализация и минимизация откликов на одну границу.

В более развернутом смысле это означает, что детектор не должен выявлять ложные границы (например, на шумы), должен правильно и не фрагментировано определять линию границы, и лишь единожды реагировать на каждую границу, чтобы избежать появления широких полос. Алгоритм детектора Канни состоит из 5 шагов. Первый шаг — сглаживание. Оно используется, когда во избежание появления ложных границ требуется уменьшить количество шумов на изображении. Для этого часто используется размытие фильтром Гаусса или каким-либо матричным фильтром размытия» [6].

$$G(x,y) = \frac{1}{2\pi\sigma^2}\epsilon^{-\frac{x^2+t^2}{2\sigma^2}}, \tag{1.6}$$

где,

(x,y) - координаты пикселя, σ - стандартное отклонение (контролирует степень размытия).

Результирующее изображение:

$$I_{blurred}(x,y) = (G * I)(x,y) = \sum_{i,j} G(i,j)I(x-i,y-j) \tag{1.7}$$

Следующие два шага — это нахождение градиентов и подавление не максимумов. Для начала находятся все градиенты яркости, для этого можно использовать, например, оператор Собеля:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \tag{1.8}$$

Градиенты по осям x и y вычисляются как:

$$\frac{\partial I}{\partial x} = G_x * I_{\text{blurred}}, \quad \frac{\partial I}{\partial y} = G_y * I_{\text{blurred}} \tag{1.9}$$

Амплитуда и направление градиента:

$$|\nabla I| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}, \quad \theta = \arctan\left(\frac{\frac{\partial I}{\partial y}}{\frac{\partial I}{\partial x}}\right) \tag{1.10}$$

«Но для того чтобы граница была четкой и понятной, она должна быть представлена тонкой линией. Поэтому границей будут являться те пиксели, в которых достигается локальный максимум градиента в направлении вектора градиента. Допустим, что почти все пиксели в градиенте имеют ориентацию вверх. Тогда значение градиента в них будет сравнено с ниже- и вышерасположенными пикселями. Те пиксели, которые имеют наибольшее значение, останутся в результирующем изображении, остальные — будут подавлены» [6]. Для получения тонких границ выполняется подавление немаксимумов:

1. Квантование направления градиента θ в одно из 4 возможных направлений:
   - $0°$ (горизонтальное),
   - $45°$,
   - $90°$ (вертикальное),
   - $135°$.
2. Для каждого пикселя (x, y) проверяется, является ли его значение $|\nabla I(x, y)|$ локальным максимумом в направлении градиента $\theta(x, y)$.

3. Если значение не является максимумом, оно подавляется (обнуляется):

$$I_{\text{thin}}(x,y) = \begin{cases} |\nabla I(x,y)|, & \text{если } |\nabla I(x,y)| \geq \text{значения соседей вдоль } \theta(x,y) \\ 0, & \text{иначе} \end{cases}$$

где,

— I_{thin} - изображение с подавленными немаксимумами

— Соседи определяются по квантованному направлению θ

«И последние этапы — это двойная пороговая фильтрация и трассировка области неоднозначности. На данном шаге производится еще одна фильтрация ложных границ. В детекторе границ Канни используется два порога: нижний и верхний. Пиксель, значение которого выше верхней границы, принимает максимальное значение, т. е. контур считается достоверным. Если значение пикселя не достигает нижнего порога — пиксель подавляется. Если его значение попадает в диапазон между порогами, то он принимает среднее значение, а решение о том, является ли он точкой границы, будет принято во время трассировки области неоднозначности» [12]. Для окончательного выделения границ применяется метод двойного порога:

1. Задаются два пороговых значения:

   — T_{high} - верхний порог (сильные границы),
   — T_{low} - нижний порог (слабые границы);

2. Классификация пикселей:

$$I_{\text{edges}}(x,y) = \begin{cases} \text{strong}, & \text{если } |\nabla I(x,y)| \geq T_{\text{high}} \\ \text{weak}, & \text{если } T_{\text{low}} \leq |\nabla I(x,y)| < T_{\text{high}} \\ 0, & \text{иначе} \end{cases}$$

где,

— I_{edges} - итоговое изображение границ,
— Рекомендуемое соотношение порогов: $T_{\text{high}} = 2 \cdot T_{\text{low}}$,
— Типичные значения: $T_{\text{low}} = 0.05 \cdot \max(|\nabla I|)$, $T_{\text{high}} = 0.15 \cdot \max(|\nabla I|)$;

3. Сохранение только значимых слабых границ:

   — Слабые границы (weak) сохраняются только если они соединены с сильными (strong) через цепочку соседних пикселей (8-связность);
   — Все остальные слабые границы отбрасываются;

Как можем наблюдать в задачах обработки изображений, процесс трассировки границ основывается на анализе пикселей со средними значениями интенсивности. Алгоритм работает по принципу последовательного преобразования таких пикселей в зависимости от их окружения.

Пиксель, получивший среднее значение, подвергается проверке на соседство с достоверными граничными элементами. В случае обнаружения контакта с четким контуром, значение данного пикселя повышается до максимального, что приводит к его включению в состав границы. При отсутствии же значимого соседства с достоверными граничными элементами происходит подавление такого пикселя, исключая его из дальнейшего анализа.

Рисунок 1.3 - Результаты работы фильтра Канни

1.2.3 Получение фона

Для того чтобы на любой итерации движения робота быстро получать его актуальную позицию, используя вычитание модели заднего плана, нам необходима эта модель. Следовательно, первым делом, необходимо понять, как получить задний план. Будем использовать следующий метод:

— Извлечение контуров объектов из изображения.

— Удаление автомобиля (самого маленького объекта) из маски.

— Создание модели заднего плана с заполнением пустых мест.

— Нахождение области лабиринта и преобразование её в подходящий формат.

Исходное изображение лабиринта с камеры конвертируем в градации серого с использованием функции cv2.cvtColor. Это упрощает анализ, убирая информацию о цвете.

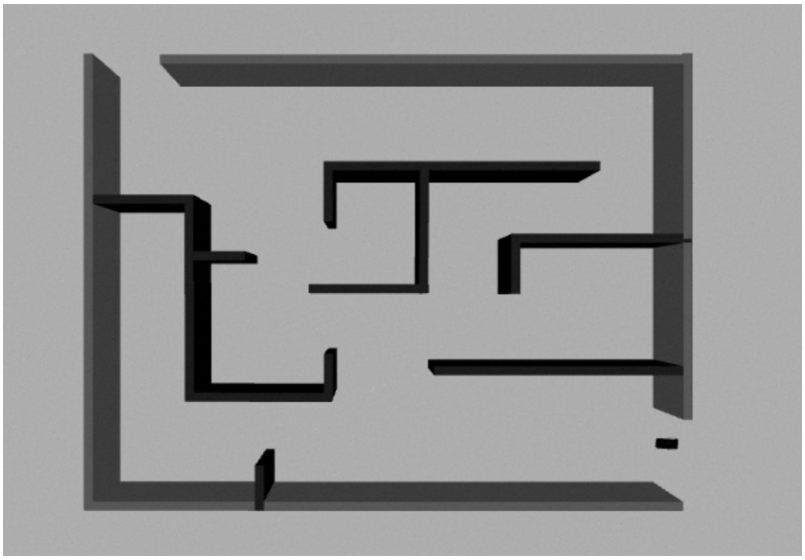


Рисунок 1.4 - Изображение лабиринта с камеры в градациях серого

Далее, используя алгоритм детекции краев Джона Ф. Канни, определим края стенок и робота. Воспользуемся функцией cv2.Canny, передав в нее изображение лабиринта в градациях серого.

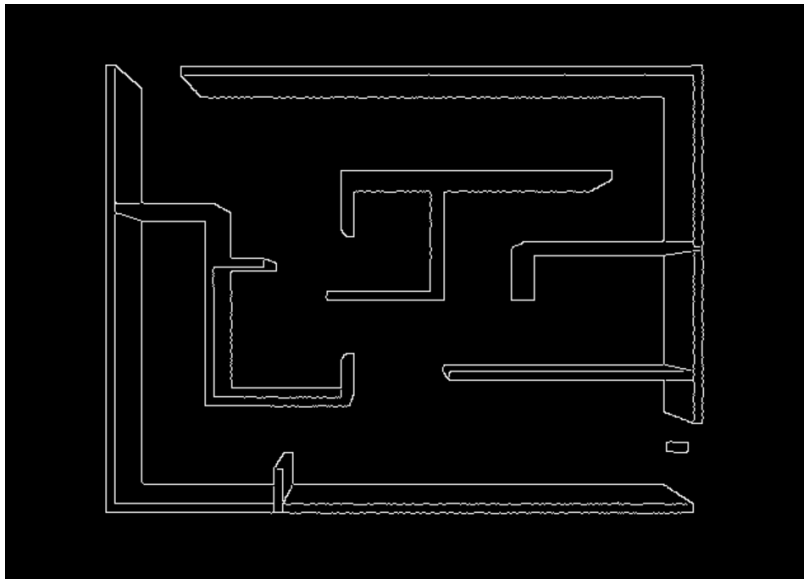


Рисунок 1.5 - Детекция краев лабиринта по алгоритму Канни

Нахождение контуров: функция cv2.findContours извлекает внешние контуры объектов на изображении.

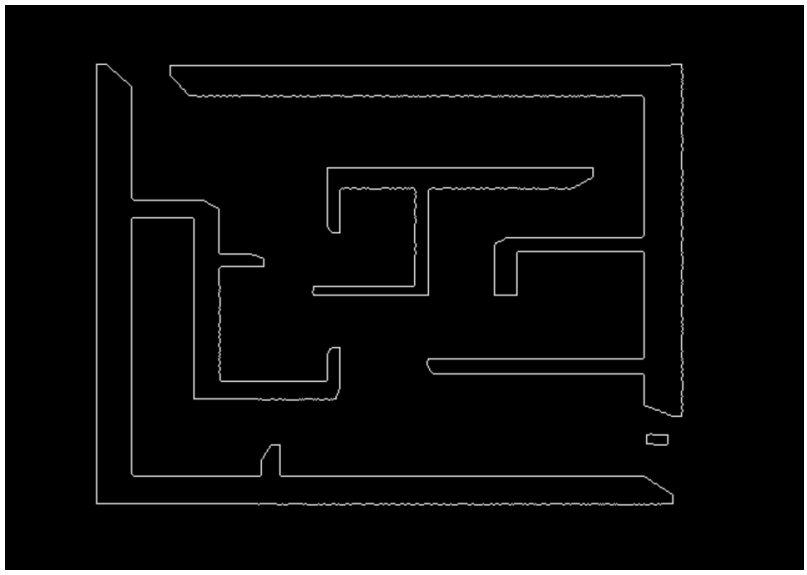


Рисунок 1.6 - Внешние контуры на изображении

Построение маски: На основе найденных контуров создаётся маска, где каждая область заполняется белым цветом (значение 255).

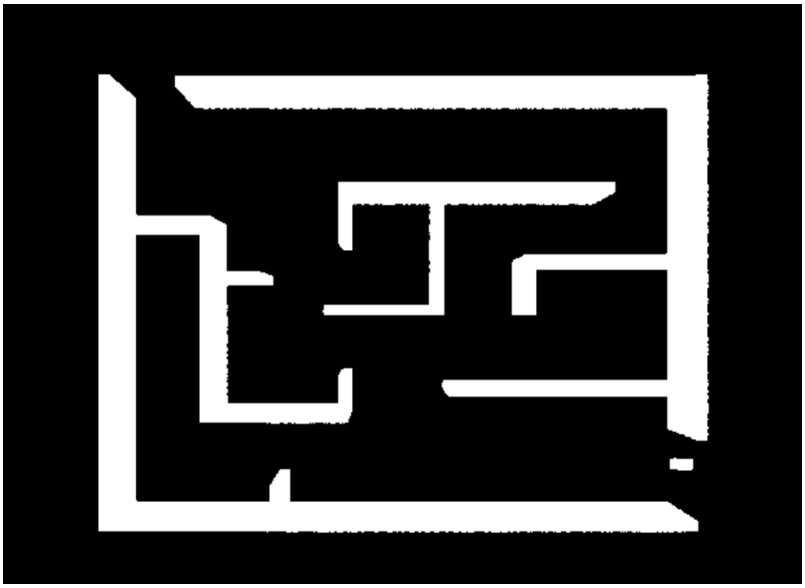


Рисунок 1.7 - Маска изображения

Далее, используя внешние контуры на изображении, можем найти самый маленький элемент на картинке. Будем идти по всем контурам, высчитывать их площадь и искать наименьшую – это и будет робот. Имея контуры модели робота, рисуем битовую маску, применив которую к картинке с камеры, модель робота будет удалена.

Рисунок 1.8 - Битовая маска с удаленной моделью робота

Применив логическое ’и’ между битовой маской и изображением с камеры, получим следующее изображение.

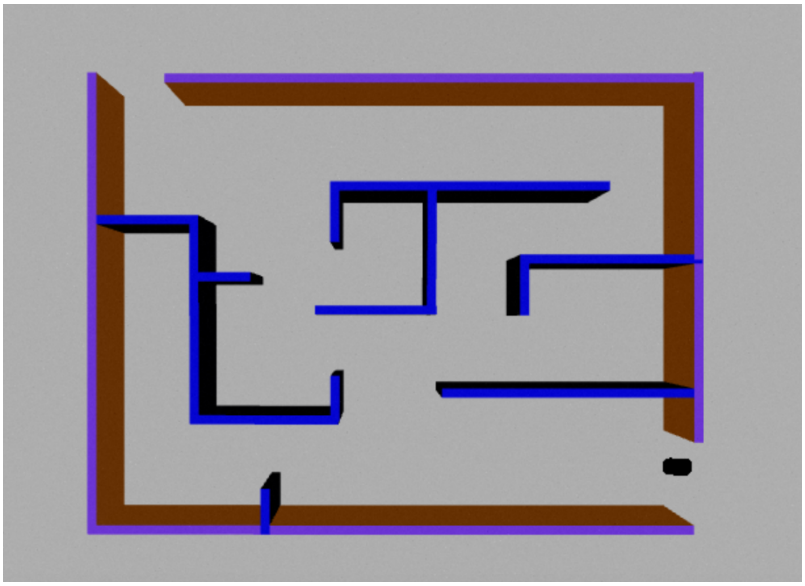


Рисунок 1.9 - Изображение лабиринта с удаленной моделью робота

После чего остается заполнить удаленное пространство цветом фона.

1.3 Методы картирования

Следующий вопрос, на который стоит ответить, - это: есть ли у меня карта лабиринта? Для того чтобы ответить на этот вопрос, будем использовать картирование. Картирование - это представление пространства в виде карты, которую способна понять машина. Применяется в робототехнике, навигации и картографии, чтобы понимать, как устроено окружение, и принимать решения на основе этих данных. Существует 2 вида картирования: картирование с сеткой и топологическое картирование.

Grid Mapping – метод картирования, когда пространство делится на ячейки, как лист бумаги в сетке. Каждая ячейка показывает, что находится в этом месте: свободно, занято (препятствие) или неизвестно. Этот метод даёт подробную карту и часто используется в робототехнике.

Grid mapping активно используется в робототехнике для создания карт и навигации в замкнутых или неопределённых пространствах, таких как:

— SLAM (Simultaneous Localization and Mapping): В роботе с применением SLAM, карты часто строятся с использованием сеток, где каждая клетка может быть

оценена как пустая, занятая или неопределённая.

— Навигация автономных транспортных средств: Автономные автомобили и дроновые системы используют сетки для планирования маршрутов, избегания препятствий и предотвращения столкновений.

Topological Mapping (Топологическое картирование) — это подход, который моделирует пространство, как сеть узлов, соединённых рёбрами, где каждый узел представляет собой абстрактное место или состояние в пространстве, а рёбра отражают возможные переходы между этими состояниями. Топологическое картирование не хранит информацию о каждой точке пространства, а скорее показывает, как можно перейти из одного состояния в другое.

Топологическое картирование используется в сценариях, где важно абстрактное представление пространства:

— Навигация в больших или динамических пространствах. Например, в больших зданиях, таких как аэропорты или торговые центры.

— Роботы и автономные системы. Роботы, работающие в незнакомых или динамических средах, топологическое картирование является эффективным способом представления пространства.

— Игры и симуляции. В некоторых видеоиграх или симуляциях используется топологическое картирование для моделирования переходов между локациями, где важна сама структура мира, а не его детализированное отображение.

Разберем наш случай и проанализируем, какой лучше вид картирования использовать.

1. Имеем статическое окружение в виде лабиринта (необходимо лишь один раз провести процесс картирования).

2. Лабиринт с условием одного входы и одного выхода.

3. Может иметь несколько путей от входа до выхода.

4. Стены являются не проходимыми.

Имеем, что лабиринт очень удобно представить в виде графа, то есть будем использовать топологическое картирование.

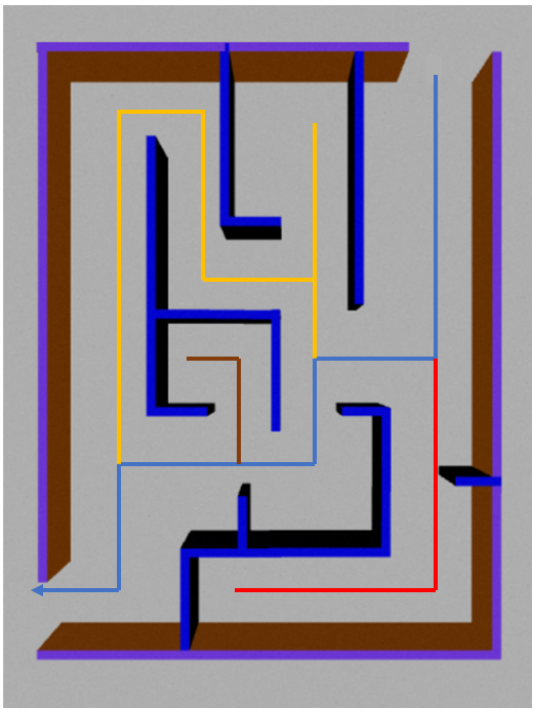


Рисунок 1.10 - Топологическое картирование на лабиринте

1.3.1 Граф

Граф — это математическая структура, используемая для моделирования объектов и их связей. Он состоит из двух основных элементов:

1. Вершины. Основные объекты, которые граф моделирует. Например, города в сети дорог или точки пересечения путей в лабиринте.

2. Рёбра. Связи между вершинами. Могут быть: ориентированными — связь имеет направление, неориентированными — связь не имеет направления.

Графы могут быть:

1. Взвешенными — рёбрам присваиваются веса. Например, расстояния или стоимость.

2. Не взвешенными — все рёбра равнозначны.

Примеры использования графов:

1. Транспортные сети: города и дороги между ними.

2. Социальные сети: пользователи и их связи (дружба, подписки)

3. Навигация: робототехника, анализ маршрутов.

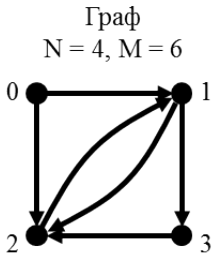
4. Вычислительные сети: Узлы сети и соединения между ними.

Представление графа в памяти компьютера.

Существуют два основных способа представления графов в памяти компьютера:

1. Матрица смежности. Матрица смежности — это двумерный массив n×n, где n — количество вершин графа. Каждый элемент матрицы A[i][j] показывает наличие (или вес) ребра между вершинами i и j.

2. Список смежности — это структура данных, где каждая вершина графа хранит список соседей, с которыми она соединена. Если граф взвешенный, в списке также указываются веса рёбер.



Граф
N = 4, M = 6

Список рёбер

(0; 1)
(0; 2)
(1; 2)
(1; 3)
(2; 1)
(3; 2)

Матрица смежности

	0	**1**	**2**	**3**
0	0	1	1	0
1	0	0	1	1
2	0	1	0	0
3	0	0	1	0

Списки смежности

0	1	2
1	2	3
2	1	
3	2	

Рисунок 1.11 - Пример матрицы смежности и списка смежности

В качестве представления графа в памяти компьютера будем использовать список смежности по следующим причинам:

1. Хранение меньшего количества информации.

2. Возможность быстрее обходить всех соседей интересующей нас вершины.

1.3.2 Интересные точки

Рассмотрим следующую ситуацию: пусть у нас есть путь из одной точки в другую, который является прямым или по-диагонали. Проблема заключается в том, что информация, которая находится между двумя точками, никак не используется.

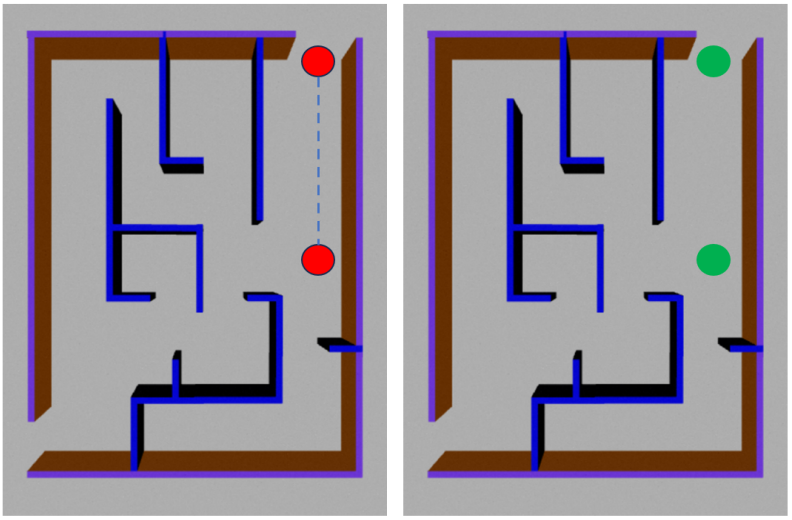


Рисунок 1.12 - Выделение интересных точек

Для решения этой проблемы будем использовать выделение так называемых интересных точек. Интересная точка - это некоторая область, в которой мы делаем решение о дальнейшем движении робота. Можем выделить 4 основные "интересные"точки.

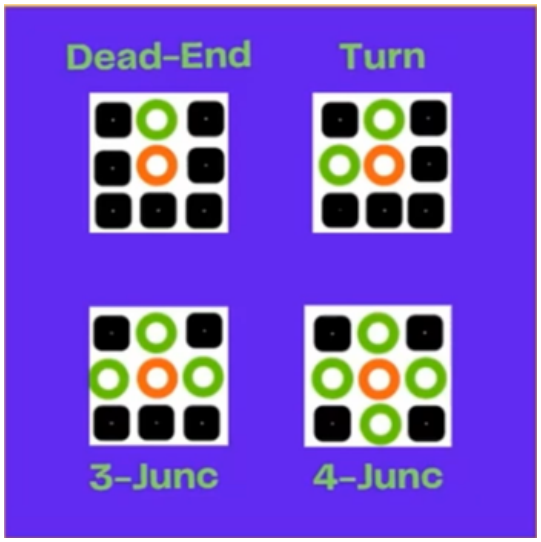


Рисунок 1.13 - Типы интересных точек

Имеем: тупик, поворот, тройной перекресток, четверной перекресток. Выделив виды интересных точек, можем приступать к формированию графа.

1.3.3 Обработка изображения

Предварительно для того чтобы произвести картирование лабиринта, необходимо обработать маску, полученную на итерации локализации. Применим функцию thinning из пакета OpenCV2 для того чтобы построить скелет лабиринта. В функции thinning используется алгоритм Zhang-Suen для утонения. Алгоритм постепенно удаляет внешние пиксели бинарного изображения, пока не останется только "скелет"объекта толщиной в 1 пиксель. Работает за несколько проходов, пока на очередном проходе еще есть что удалять. Для каждого пикселя проверяем его 8 соседей:

p_9	p_2	p_3
p_8	p_1	p_4
p_7	p_6	p_5

Пиксель p_1 может быть удален, если выполняются условия:

1. $2 \leq B(p_1) \leq 6$, где $B(p_1) = \sum_{i=2}^{9} p_i$ - количество соседей-объектов

2. $A(p_1) = 1$, где $A(p_1)$ - число переходов $0 \to 1$ при обходе соседей:

$$A(p_1) = \sum_{i=2}^{9} \mathbb{I}(p_i = 0 \wedge p_{i+1} = 1), \qquad p_9 = p_2 \tag{1.11}$$

Алгоритм чередует два типа проверок:

1. Первый проход:

$$P2 \cdot P4 \cdot P6 = 0, \tag{1.12}$$

$$P4 \cdot P6 \cdot P8 = 0 \tag{1.13}$$

2. Второй проход:

$$P2 \cdot P4 \cdot P8 = 0, \tag{1.14}$$

$$P2 \cdot P6 \cdot P8 = 0 \tag{1.15}$$

Алгоритм завершается, когда на обоих проходах подряд не было удалено ни одного пикселя.

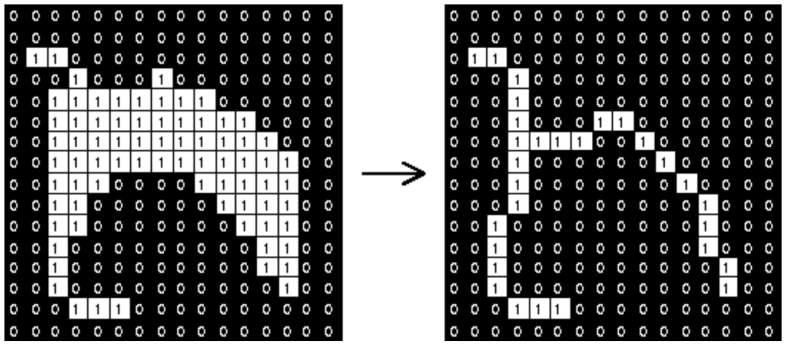


Рисунок 1.14 - Работа функции thinning

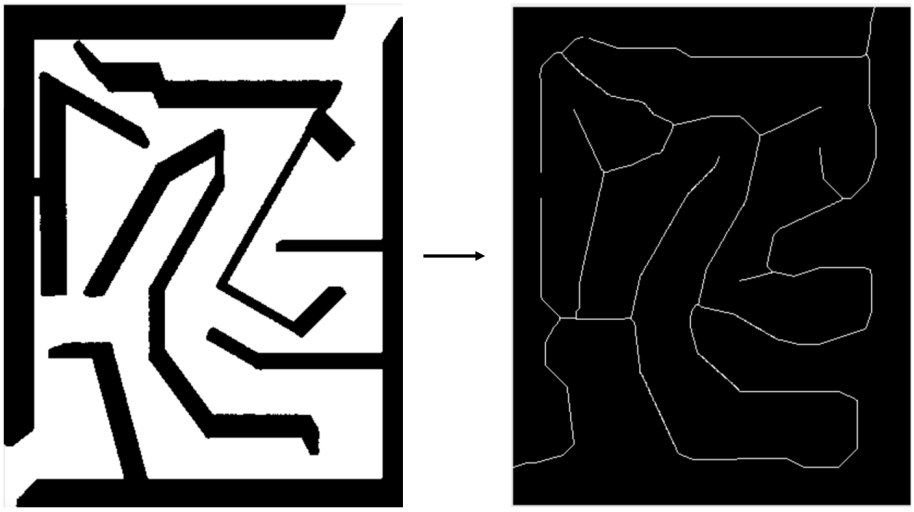


Рисунок 1.15 - Выделение скелета лабиринта

Для улучшения качества скелета произведем расширение линий и повторное утонение. Это поможет исправить артефакты, разрывы и нечеткости, которые могут возникнуть в процессе начального утонения или из-за шума в исходном изображении.

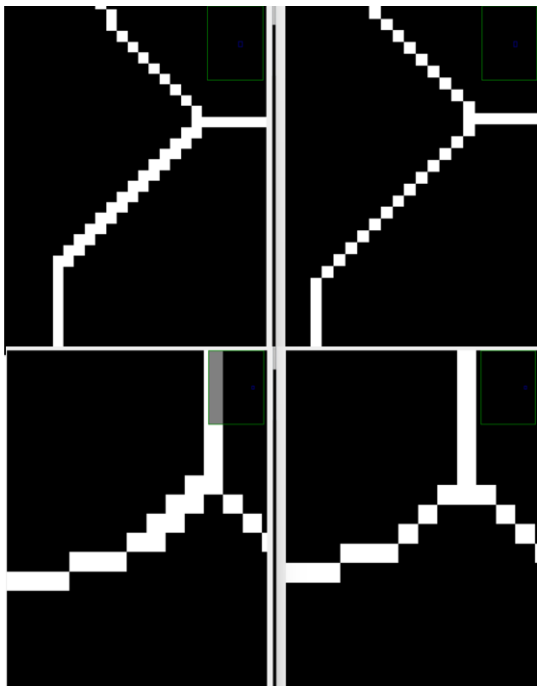


Рисунок 1.16 - Результаты повторного утонения

Как можем заметить, линии стали более чёткими, а также исчезли шумы.

1.3.4 Один проход

На данном этапе, имея скелет лабиринта и типы интересных точек, будем составлять граф. Будем делать это с помощью одного прохода по матрице пикселей. Проходя по матрице, идентифицируем каждый пиксель как ту или иную интересную точку; если он белого цвета, то есть относится к лабиринту. Для каждого пикселя вычисляем значения интенсивностей всех соседних позиций, а именно восемь направлений: верхний-левый, верхний, верхний-правый, левый, правый, нижний-левый, нижний, нижний-правый. Если соседний пиксель находится за пределами лабиринта (например, в случае верхней строки и верхнего-левого пикселя), ему присваивается значение 0. Вычисляем сумму всех значений окружающих пикселей, после чего анализируем на тип интересной точки. Если имеем сумму пикселей 1, значит — это тупик. Сумма пикселей 2 означает, что это поворот и мы имеем два выхода из данной точки. Сумма 3 и 4 показывает нам то, что данная точка тройной или четверной перекресток соответственно. В результате получим взвешенный граф, вершины которого содержат информацию о том, каким видом интересной точки является пиксель. После этого необходимо добавить рассматриваемый пиксель как вершину в граф и соединить её с соседями. Для этого будем сканировать пиксели в четырёх направлениях, а именно: слева, слева-сверху, сверху, справа-сверху.

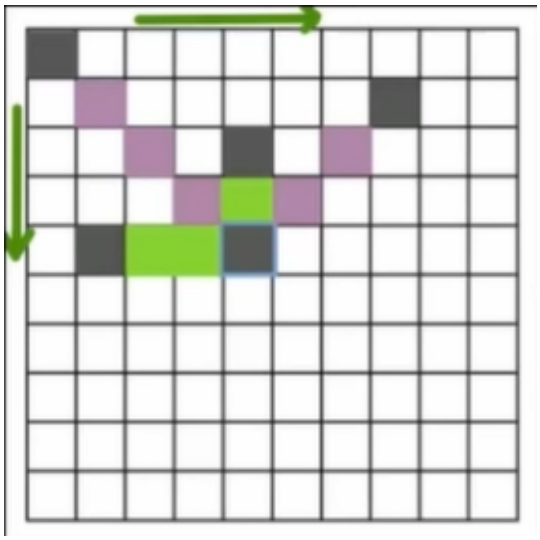


Рисунок 1.17 - Направления поиска соседей

Если в одном из этих четырех направлений мы нашли белый пиксель, то добавляем его в граф и образуем связь с текущей вершиной, и так рекурсивно по каждому из четырех направлений.

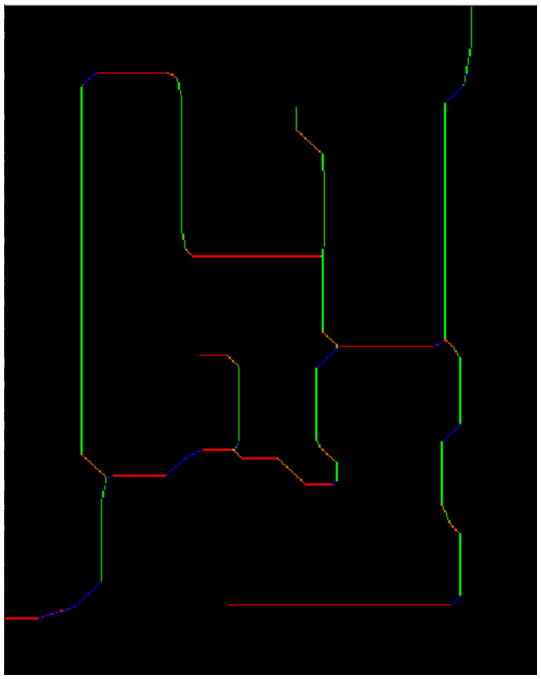


Рисунок 1.18 - Выделение каждого направления цветом

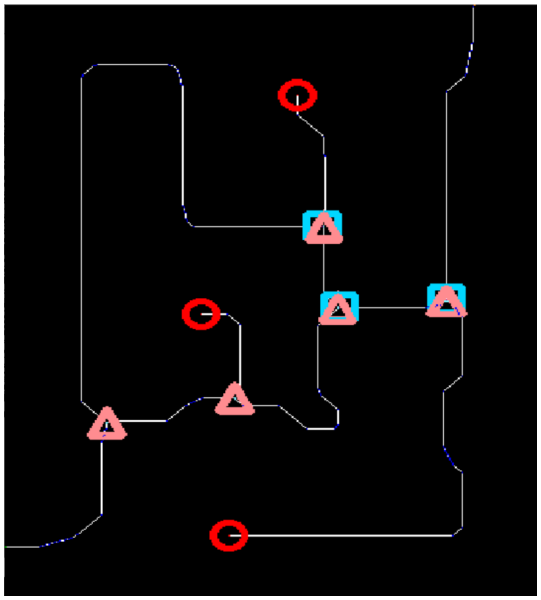


Рисунок 1.19 - Выделение интересных точек на скелете

Теперь, после того как мы произвели полный процесс картирования лабиринта и представили его в виде взвешенного графа, можем приступить к ответу

на следующий вопрос, а именно, поиск оптимального пути.

1.4 Методы поиска оптимального пути

Ответим на следующий вопрос, а именно: какой самый короткий путь к цели? Для этого будем использовать алгоритмы для поиска пути в графе. «Алгоритмы поиска пути используют структуру графа и веса его рёбер, чтобы найти наиболее оптимальный путь между двумя вершинами. Понятие «оптимальный» путь меняется в зависимости от задачи. В одном случае нам нужно найти самый короткий путь, в другом — самый дешевый или путь с наименьшим количеством препятствий. Также рассмотрим еще один известный алгоритм поиска путей в графе, а именно поиск в глубину. Проанализируем все три алгоритма, разберём, как они работают, и протестируем на нашем лабиринте» [1].

1.4.1 Поиск в глубину

«DFS реализует концепцию «погружайся глубже, головой вперед». Суть алгоритма заключается в том, что мы начинаем движение от начальной вершины в определённом направлении, пока не достигнем конца пути. Придя в конечную точку, мы проверяем, является ли она пунктом назначения, и если нет, то мы возвращаемся в начальную точку и выбираем другой маршрут» [7]. Алгоритм может быть реализован двумя способами:

1. Рекурсивный, то есть с использованием структуры данных стек.
2. Итеративный, то есть с явным использованием структуры данных "стек".

Сложность алгоритма:

— Временная сложность: $O(V+E)$, где V — количество вершин, E — количество рёбер. Каждая вершина и каждое ребро обрабатываются один раз.

— Пространственная сложность:

  – В рекурсивной реализации — $O(V)$ из-за стека вызовов.

– В итеративной реализации — $O(V)$ из-за явного стека.

Применение:

— Определение связности графа: DFS позволяет проверить, связен ли граф, посетив все вершины, достижимые из стартовой вершины.

— Поиск циклов: С помощью DFS можно определить, содержит ли граф циклы.

— Топологическая сортировка: В ориентированных ациклических графах DFS используется для построения упорядоченного списка вершин.

— Решение лабиринтов: DFS может применяться для нахождения путей в лабиринтах и подобных задачах.

Преимущества и недостатки:

Преимущества:

— Простота реализации.

— Эффективен для графов, где важна глубина исследования.

Недостатки:

— Может потребовать значительных ресурсов памяти в случае глубоких рекурсий.

— Подвержен зацикливанию, если не использовать проверку посещенных вершин.

1.4.2 Алгоритм Дейкстры

«Алгоритм Дейкстры — это классический алгоритм поиска пути, который был разработан голландским ученым Эдсгером Дейкстрой в 1959 году. Этот алгоритм используется для поиска кратчайшего пути в взвешенном графе от одной вершины (обозначим её как начальную) до всех остальных вершин.

Принцип работы:

Алгоритм Дейкстры начинается с установки начальной вершины и работы от этой точки. Он работает по принципу «жадного» алгоритма, что означает, что на каждом шаге он стремится минимизировать текущую общую стоимость пути.

Сначала инициализируются два множества:

— Множество, содержащее уже обработанные вершины (изначально пустое).

— Множество, содержащее все остальные вершины графа (изначально содержит все вершины графа).

Также каждой вершине графа присваивается вес, который представляет минимальную известную стоимость пути от начальной вершины до данной. Для начальной вершины этот вес равен 0, для всех остальных вершин — бесконечность.

На каждом шаге алгоритм выбирает вершину из непосещенного множества с наименьшим весом, перемещает эту вершину в множество посещенных вершин и обновляет веса всех соседей выбранной вершины. Вес соседа обновляется, если через выбранную вершину можно добраться до этого соседа с меньшей стоимостью.

Процесс продолжается, пока не будут посещены все вершины или пока мы не найдем путь до конечной вершины (если она задана)» [1].

Сложность алгоритма:

— Временная сложность: $O((V + E)logV)$, если используется куча для реализации очереди.

— Пространственная сложность: $O(V + E)$, где V — вершины, E — рёбра, для хранения графа и расстояний.

Применение

— Навигационные системы: Определение кратчайшего пути между двумя точками.

— Сетевые маршрутизаторы: Оптимизация передачи данных.

— Оптимизация задач логистики: Планирование транспортировки.

— Решение задач в робототехнике: Построение маршрутов для мобильных роботов.

Преимущества и недостатки:

Преимущества:

— Эффективен для графов с неотрицательными весами.

— Хорошо масштабируется при использовании оптимальных структур данных.

Недостатки:

— Невозможность использования для графов с отрицательными весами (в таких задачах применяется алгоритм Беллмана-Форда).

— Медленный для больших графов с плотной структурой рёбер.

1.4.3 Алгоритм A*

Алгоритм A* представляет собой один из наиболее эффективных методов поиска оптимального пути. Данный алгоритм, разработанный Питером Хартом, Нильсом Нильсоном и Бертраном Рафаэлем в 1968 году, нашёл широкое применение в различных областях.

Принцип работы:

Алгоритм A* реализует оптимальный поиск пути через систему приоритетной обработки вершин. В основе этого механизма лежит вычисление прогнозируемого расстояния $f(x)$ для каждой вершины x, определяемого как:

$$f(x) = g(x) + h(x), \tag{1.16}$$

где,

— $g(x)$ — точная стоимость достижения вершины x из начальной точки,

— $h(x)$ — эвристическая оценка стоимости пути от текущей вершины x до целевой вершины.

Для достижения максимальной эффективности рекомендуется использовать эвристические функции, удовлетворяющие условию допустимости (не переоценивающие реальное расстояние). В типовых случаях навигации по сетке хорошо зарекомендовала себя манхэттенская метрика.

Применение:

Навигационные системы: Построение маршрутов с учётом реальных расстояний. Игровая разработка: Построение путей для NPC. Робототехника: Прокладывание оптимальных маршрутов в реальном времени. Сети: Оптимизация передачи данных с учётом расстояния и задержек.

Преимущества и недостатки:

Преимущества:

— Гарантирует нахождение оптимального пути при допустимой эвристике.

— Часто быстрее, чем алгоритм Дейкстры, благодаря использованию эвристики.

Недостатки:

— Вычислительная сложность возрастает, если эвристика недостаточно точная.

— Зависит от качества эвристической функции.

Выбор эвристики:

Эвристическая функция должна быть:

— Допустимой: $h(x) \leq$ реального расстояния до цели.

— Последовательной: $h(x) \leq d(x, y) + h(y)$, где $d(x, y)$ — стоимость пути между соседними вершинами x и y.

Для сетки с прямолинейными дорогами: манхэттенское расстояние. Для евклидового пространства: евклидово расстояние.

1.5 Выводы по второй главе

В течение данной главы были рассмотрены все методы и алгоритмы, которые могут быть применены при составлении алгоритма нахождения кратчайшего

пути. Сперва был рассмотрен метод локализации - вычитание фона. Вычитание фона - это первая итерация алгоритма, на каждом шаге необходимо понимать, где находится робот. Координаты робота необходимы для того, чтобы задавать ему направление движения и скорость, а также понимать, застрял ли робот и как находить выход из тупика. Далее был рассмотрен метод картирования, для того чтобы представлять лабиринт в памяти компьютера. В случае лабиринта наиболее эффективно использовать топологическое картирование. Почему именно топологическое картирование? На графе удобно применимы алгоритмы нахождения кратчайшего пути, которые были рассмотрены далее, а именно: DFS, Дейкстра и A*.

Таким образом, ответив на все вопросы, поставленные в начале главы, мы имеем все необходимые методы и компоненты для составления системы по решению лабиринтов.

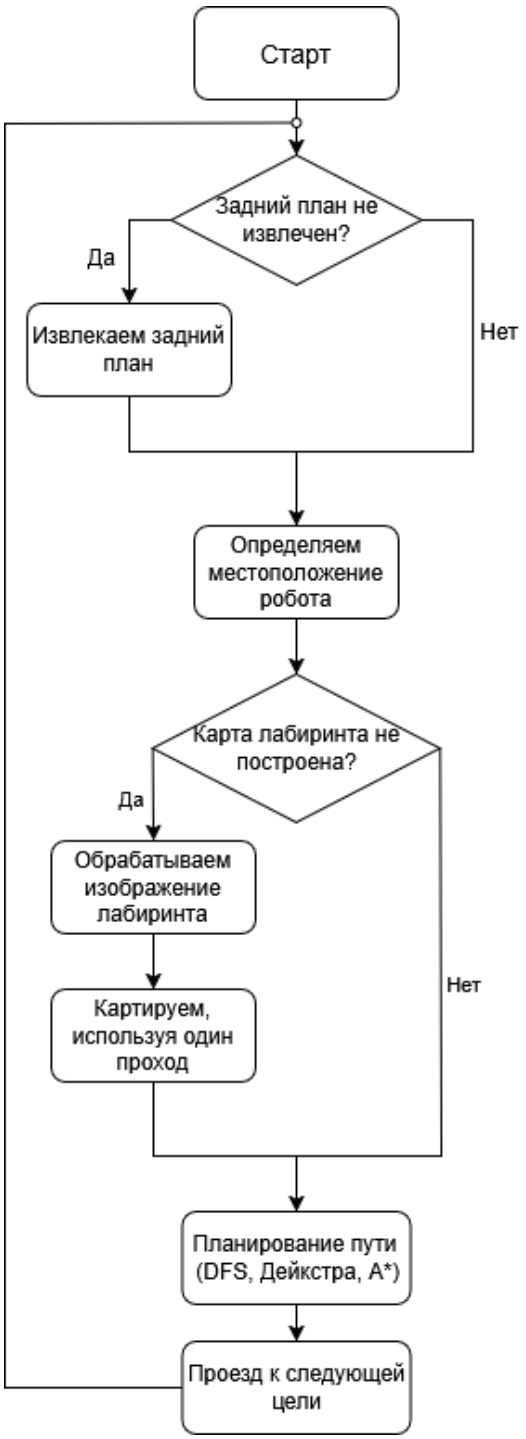


Рисунок 1.20 - Схема системы

ГЛАВА 2
РЕАЛИЗАЦИЯ АЛГОРИТМА В СРЕДЕ GAZEBO

2.1 Среда разработки

В качестве среды разработки и тестирования алгоритма я выбрал ROS2, как инструмент для манипулирования роботом, Gazebo, как симулятор робота и Python, для непосредственного программирования робота. А также библиотеку OpenCV, для обработки изображений.

«ROS2 (Robot Operating System 2) - это фреймворк для программирования роботов. ROS является «программным клеем», который даёт возможность разработчикам сосредоточиться на своей конкретной задаче. Хотя ROS не является операционной системой, он предоставляет сервисы, такие как аппаратная абстракция, низкоуровневое управление устройствами, реализация часто используемых функций, передача сообщений между процессами и управление пакетами (плагинами).

ROS спроектирована как слабо связанная система, в которой процесс, называемый узлом (node), должен отвечать за одну задачу. Узлы общаются друг с другом, используя сообщения, проходящие через логические каналы, называемые темами (topics). Каждый узел может отправлять или получать данные от другого узла, используя шаблон проектирования издатель-подписчик (publish–subscribe pattern).

Для ROS уже реализованы драйвера, позволяющие единым образом работать со многими устройствами, такими как контроллеры, GPS, камеры, лазерные дальномеры и т. п.» [14]. Некоторые из основных средств разработки и тестирования в ROS2 включают:

— ROS2 Command Line Tools предоставляет команды для управления и взаимодействия с системой ROS2. Основные команды включают создание, сборку и запуск пакетов,а также управление узлами и топиками, инструменты для отладки.

— ROS2 Launch позволяет управлять запуском и конфигурацией узлов ROS2.

Он позволяет создавать файлы с описанием узлов и связей между ними, что облегчает развертывание сложных систем ROS2.

— ROS2 Bag позволяет записывать и получать сообщения из топиков в системе ROS2. Это полезный инструмент для записи и получения данных.

— ROS2 Testing Frameworks фреймворк для модульного и интеграционного тестирования узлов и пакетов ROS2. Включают rclcppgtest для модульного тестирования узлов, основан на Google Test, а также launchtesting для проверки системы ROS2.

«Gazebo — это динамический 3D симулятор с открытым исходным кодом, который развивается Open Source Robotics Foundation и довольно тесно взаимодействует с ROS. Gazebo позволяет точно и эффективно моделировать роботов как в сложных условиях помещений, так и снаружи.

Симулятор состоит из сервера gzserver, который занимается просчетом физики, столкновений и симуляцией сенсоров. К серверу могут подсоединяться клиенты, например gzclient (для десктопа) и gzweb (для браузера). Именно они занимаются рендерингом моделей.

Все это дает возможность тестировать сложные робототехнические системы в виртуальном пространстве гораздо быстрее и без риска нанести ущерб дорогостоящим настоящим роботам.

Gazebo включен в полный установочный пакет ROS, поэтому дополнительно ничего устанавливать не нужно. Для headless конфигурации требуется gzweb» [8]. Некоторые из средств разработки и тестирования в Gazebo включают:

— Gazebo GUI: Gazebo предоставляет интерфейс с графической оболочкой, который позволяет визуализировать и управлять симуляцией. Он позволяет загружать и настраивать модели роботов, управлять физическими параметрами окружения и выполнять тестирование взаимодействия робота с окружающим миром.

— Gazebo Models: Gazebo поставляется с библиотекой моделей, которую можно использовать для создания роботов и окружающих объектов в симуляции. Это упрощает разработку и тестирование моделей роботов, а также создание сценариев симуляции.

— Gazebo Plugins: Gazebo позволяет создавать плагины, которые расширяют функциональность симулятора. Плагины могут использоваться для добавления датчиков, контроллеров или других пользовательских компонентов в симуляцию. Пишутся на C++ или Python.

Python: популярный язык программирования для разработки в области робототехники. В совокупности с ROS2 и Gazebo может использоваться для следующих задач:

— Разработка узлов ROS2. Python используется для создания узлов в ROS2, публикации и подписки на топики, а также для управления внешними системами.

— Тестирование ROS2-узлов. Инструменты для модульного тестирования узлов ROS2. Есть возможность использовать библиотеки, такие как unittest или pytest, для написания и запуска тестов, которые проверяют правильность работы узлов и обработки ошибок.

— Взаимодействие с ROS2. Доступ к ROS2 API используя библиотеку rclpy. Благодаря этой библиотеке можно создавать узлы, публиковать сообщения в топик, а также подписываться на топики.

— Управление Gazebo. Взаимодействие с Gazebo с использованием библиотеки gazeboros. Можно использовать Python для создания сценариев симуляции, миров и моделей, а также для управления роботами в Gazebo.

Python предоставляет большой набор инструментов и библиотек для разработки, тестирования и взаимодействия с ROS2, Gazebo и другими робототехническими средствами.

OpenCV (Open Source Computer Vision Library) — это библиотека для обработки изображений и компьютерного зрения, которая была разработана компанией Intel. Библиотека предоставляет большой набор инструментов и алгоритмов для работы с изображениями. OpenCV совместима со множеством языков программирования, включая Python, C++, Java. Используется в робототехнике, алгоритмах машинного зрения, а также в обработке видео и создании дополненной реальности.

— Обработка изображений: Чтение и отображение изображений в разных форматах. Преобразования формата (например, RGB в Grayscale), применение фильтров для устранения шумов, размытия и повышения резкости.

— Обработка видео: Захват потоков видео с камер, файлов или сети. Извлечение кадров, изменение разрешения и конвертация форматов, обнаружение движущихся объектов.

— Компьютерное зрение: Обнаружение и отслеживание объектов, распознавание лиц, жестов и других объектов в кадре. Построение гистограмм и анализ цвета.

— Обработка контуров и геометрии: Нахождение и анализ контуров, распознавание форм (круги, прямоугольники, треугольники), перспективные преобразования и преобразования гомографии.

— Обнаружение и отслеживание движений: Фоновая сегментация для выделения движущихся объектов, отслеживание траектории с помощью алгоритмов KLT или Optical Flow.

— Дополненная реальность: Встраивание виртуальных объектов в реальные сцены, распознавание маркеров и трёхмерные преобразования.

Преимущества OpenCV

— Масштабируемость. OpenCV эффективно обрабатывает как небольшие изображения, так и потоковые видео в реальном времени.

— Кроссплатформенность. Библиотека поддерживается на Windows, Linux, macOS и мобильных операционных системах (Android и iOS).

— Открытый исходный код. Доступность и гибкость для модификаций под различные задачи.

— Интеграция с популярными библиотеками. OpenCV совместима с NumPy, TensorFlow, PyTorch и другими инструментами для машинного обучения.

Применение в дипломной работе

В данной дипломной работе OpenCV используется для создания системы навигации мобильного робота, включающей:

— Обнаружение препятствий. Использование алгоритмов обработки изображений для выделения границ и анализа препятствий в поле зрения камеры робота.

— Навигация по маршруту: Распознавание дорожных разметок или ориентиров, на основании которых робот корректирует своё движение.

— Отслеживание положения робота: Камера фиксирует положение робота относительно объектов, используя алгоритмы распознавания форм и цвета.

2.1.1 Модель робота

ROS2 предоставляет удобный инструмент для создания своей модели робота. Для тестирования нашего алгоритма составим модель дифференциального колесного робота. Дифференциальный колесный робот — это мобильный робот, движение которого основано на двух колесах с отдельным приводом, расположенных по обе стороны корпуса робота. Таким образом, он может менять свое направление, изменяя относительную скорость вращения колес, и, следовательно, не требует дополнительного рулевого управления.

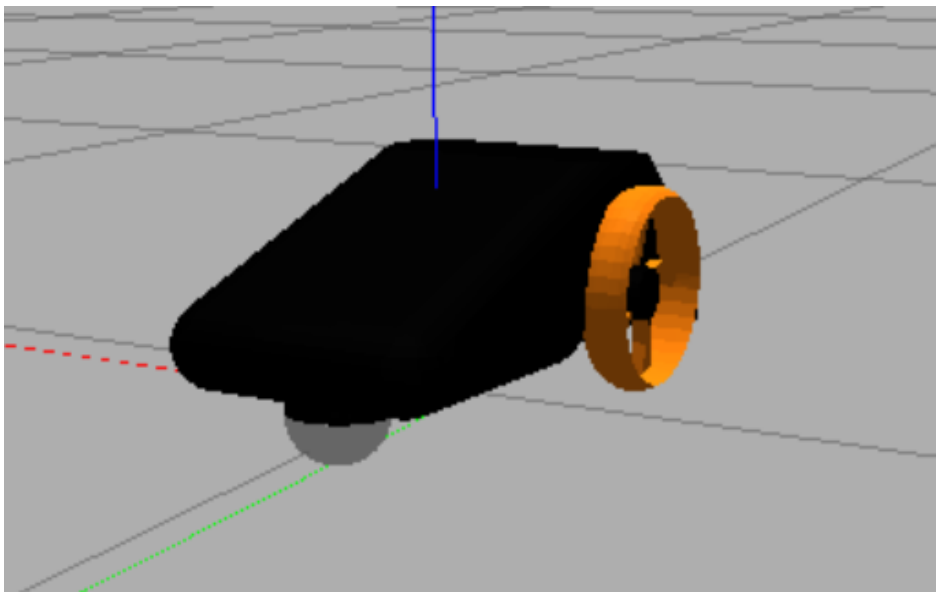


Рисунок 2.1 - Модель "робот-тележка"

2.1.2 Описание модели робота и камеры в Gazebo

Разработка имитационной модели робота и камеры в ROS начинается с проектирования его структуры в формате URDF (Unified Robot Description Format), который включает описание звеньев, соединений, массо-инерционных характеристик и других физических свойств, а также трансмиссии.

URDF — это XML-формат, используемый для описания всех аспектов робототехнических систем в ROS. Он описывает физические свойства модели, включая размеры, форму, материалы, а также динамические и кинематические аспекты. URDF обычно используется для описания статических объектов.

XACRO - расширение URDF. Предоставляет возможность создавать модульные описания, используя макросы XML, для упрощения и повторного использования кода. XACRO улучшает управляемость и поддержку сложных файлов URDF, что делает его идеальным для больших робототехнических систем.

SDF — это формат, разработанный специально для описания объектов и сред, используемых в симуляционных программах типа Gazebo. В отличие от URDF, SDF поддерживает описание не только роботов, но и целых сред, динамических свойств объектов, освещения, физики и других аспектов, что делает его особенно подходящим для комплексных симуляций.

Основные блоки URDF, которые использовались для создания модели робота:

— База робота (base)

```
<link name="base">
  <inertial>
    <mass value="2" />
    <inertia ixx="0.00355" iyy="0.01265" izz="0.013832" ... />
  </inertial>
  <visual>
    <geometry><mesh filename="package://maze_bot/meshes/base_link.stl"/></
   geometry>
  </visual>
  <collision>
    <geometry><mesh filename="package://maze_bot/meshes/base_link.stl"/></
   geometry>
  </collision>
</link>

```

Листинг 2.1 Блок base

Задает основную платформу робота с массой 2 кг. Использует STL-модель для визуализации и коллизий. Указывает тензор инерции.

— Колеса wheel-left

```
<link name="wheel_left">
  <inertial>
    <mass value="0.75"/>
    <inertia ixx="0.001444223" ... />
  </inertial>
  <visual>
    <geometry><mesh filename="package://maze_bot/meshes/wheel_l.stl"/></
   geometry>
  </visual>
</link>

<joint name="wheel_left_joint" type="continuous">
  <origin xyz="0.09 -0.1 0.04" rpy="0 0.0 3.14"/>
  <parent link="base"/>
  <child link="wheel_left"/>
  <axis xyz="0.0 1 0"/>
</joint>

```

Листинг 2.2 Блоки описывающие колеса

Описывает колеса с массой 0.75 кг. с моментом инерции. Параметр continuous позволяет колесу вращаться бесконечно. Ось вращения axis xyz="0.0 1 0"(Y-ось для левого колеса).

— Пассивное колесо caster

```
<link name="caster">
  <inertial><mass value="0.75"/></inertial>
  <visual><mesh filename="package://maze_bot/meshes/front_caster.stl"/></
   visual>
</link>
<joint name="caster_joint" type="continuous">
  <axis xyz="0 0 1" />
</joint>

```

Листинг 2.3 Блок пассивного колеса

Описывает свободно вращающееся колесо (типа "омни-колеса"), которое нужно для балансировки робота.

— Камера camera-link

```
<link name="camera_link"/>

<joint name="camera_joint" type="fixed">
  <origin xyz="-0.16 0.0 0.08" rpy="0 0.0 3.14"/>
```

```
  <parent link="base"/>
  <child link="camera_link"/>
</joint>

```

Листинг 2.4 Python example

Фиксирует камеру на корпусе с смещением (-0.16, 0.0, 0.08). Поворот rpy="00.0 3.14"разворачивает камеру на 180° по оси Z.

2.1.3 rqt graph

ROS представляет работу робота и его управление в виде взаимодействия некоторых компонентов. Данные компоненты делятся на группы: Нода(Node), Сообщение(Message) и Топик (Topic).

— Node. Как можем увидеть в архитектуре ROS, нода представляет собой минимальную исполняемую единицу системы. По своей сути, нода - это отдельный процесс, выполняющий конкретную задачу. ROS рекомендует выделять каждую функциональную задачу в отдельную ноду, что повышает модульность системы и упрощает повторное использование кода.

При запуске нода выполняет регистрацию в мастер-узле, передавая информацию о своем имени и типах обрабатываемых сообщений. После успешной регистрации нода получает возможность взаимодействовать с другими компонентами системы, осуществляя прием и передачу данных.

— Topic. Топик в ROS реализует механизм обмена сообщениями по модели "издатель-подписчик". Как можем наблюдать, этот подход аналогичен тематическому общению, где:

Издатель (publisher) сначала регистрирует топик в мастер-узле, после чего получает возможность публиковать в него сообщения. Подписчики (subscribers), заинтересованные в получении этих данных, через мастер-узел получают информацию о местоположении топика и устанавливают прямое соединение с издателем.

Такой механизм обеспечивает декомпозицию системы и позволяет нодам взаимодействовать друг с другом, сохраняя минимальную связанность компо-

нентов. Основное преимущество данного подхода заключается в том, что участники обмена сообщениями не требуют явного знания друг о друге, взаимодействуя исключительно через топики.

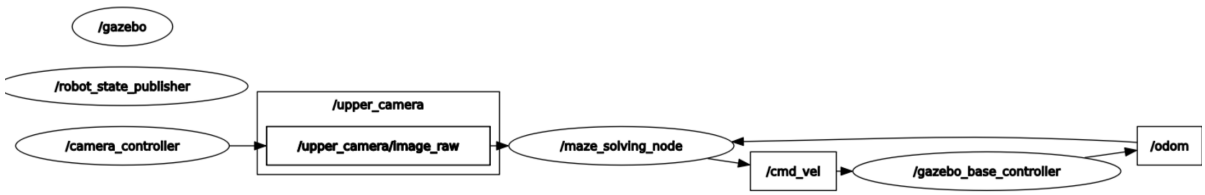


Рисунок 2.2 - rqt graph

2.2 Тестирование на лабиринте

Проведем тестирование системы на двух лабиринтах. Рассмотрим, какой алгоритм поиска кратчайшего пути на практике даёт наилучший результат. Имеем два лабиринта: один с прямыми стенками, другой с косыми, для того чтобы лучше выявить минусы алгоритма.

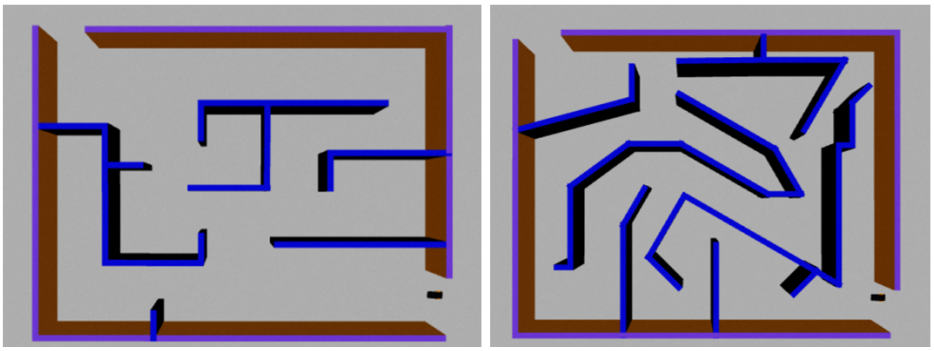


Рисунок 2.3 - Лабиринты для тестирования

Рассмотрим работу алгоритмов поиска пути на первом лабиринте.

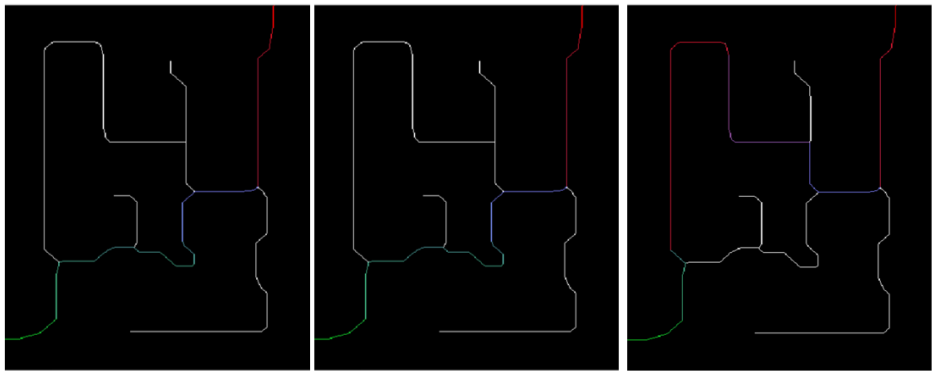


Рисунок 2.4 - Результаты работы Дейкстры, A*, DFS на 1 лабиринте

Можем заметить, что DFS выбрал не самый оптимальный путь. Стоимость или длина данного пути равна 1033, тогда как Дейкстра и A* нашли путь со стоимостью равной 671. Однако эффективность алгоритма A* раскрывается, если мы посмотрим на количество посещённых вершин. Алгоритм Дейкстры посетил 161 вершину, тогда как A* 117, что говорит о прямой зависимости с временем исполнения алгоритма. Проведем такое же исследование на втором лабиринте и получим следующие результаты: алгоритм Дейксты посетил 805 вершин, а алгоритм A* 487.

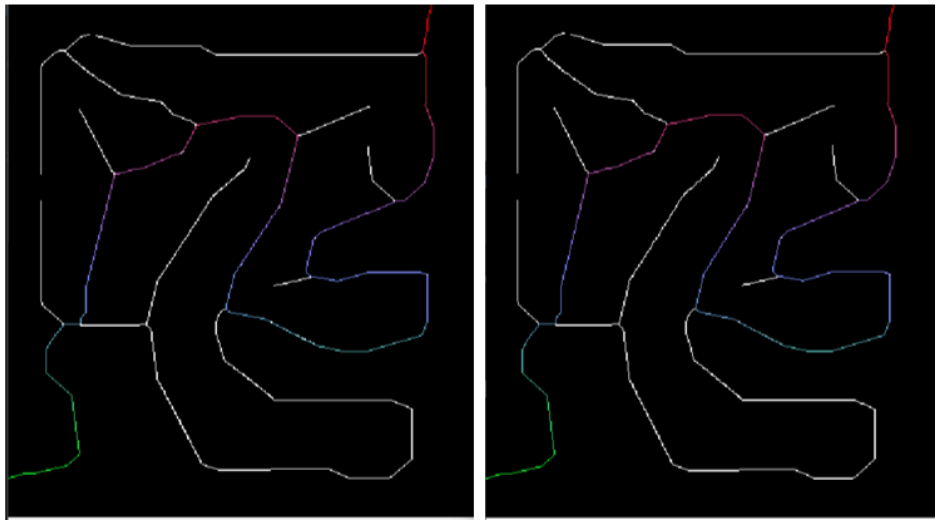


Рисунок 2.5 - Результаты работы Дейкстры, A* на 2 лабиринте

2.3 Выводы по третьей главе

Background Subtraction в ROS2 и Gazebo с использованием OpenCV хорошо работает для выделения движущихся объектов в статичной среде. В совокупности с алгоритмами поиска пути образуется эффективная система для решения лабиринтов. Описанная и реализованная в работе система имеет место быть. Как можем увидеть, мы эффективно обработали два лабиринта: один с ровными стенками, другой с кривыми. Смогли применить на них известные алгоритмы поиска пути. Сравнили их эффективность на практике и запустили робота.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы была разработана и успешно протестирована система навигации мобильного робота в условиях лабиринта, где в качестве источника информации выступала камера сверху. Основной целью исследования являлось создание алгоритма, способного не только исследовать лабиринт, но и строить оптимальный маршрут движения, исключая тупиковые участки. Поставленные задачи были выполнены в полном объеме, что подтверждается следующими результатами:

1. Анализ предметной области позволил выявить наиболее эффективные алгоритмы навигации (A*, Дейкстры, DFS) и определить их применимость для решения задач подобного типа. Особое внимание было уделено соревнованиям Micromouse, где подобные алгоритмы успешно используются на практике.

2. Реализация системы в среде ROS2 с использованием Gazebo для моделирования и OpenCV для обработки изображений подтвердила работоспособность предложенного подхода. Особое внимание было уделено вопросам локализации робота и построения карты лабиринта.

3. Тестирование алгоритма на различных типах лабиринтов показало: высокую эффективность A* по сравнению с другими алгоритмами поиска пути, а также устойчивость к изменениям конфигурации лабиринта.

4. Практическая значимость работы заключается в возможности применения разработанных решений в складской логистике, системах автоматического мониторинга и других областях, где требуется автономная навигация в ограниченном пространстве.

Проведенное исследование подтвердило гипотезу о возможности создания эффективной системы навигации на основе комбинации классических алгоритмов поиска пути и современных методов компьютерного зрения. Разработанное решение отличается относительной простотой реализации при сохранении высокой надежности работы, что делает его перспективным для практического применения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритмы поиска пути: Алгоритм дейкстры и A* [Электронный ресурс] // Хабр (компания OTUS). – 2023. – Режим доступа: https://habr.com/ru/companies/otus/articles/748470/. – Дата доступа: 12.04.2025.

2. Буласова В. Г., Беспятый Г. Ю., Белай В. Е. Обзор алгоритмов машинного обучения для позиционирования мобильной робототехнической системы //Завалишинские чтения 22. – 2022. – С. 75-78.

3. Введение в автономные мобильные роботы / Р. Зигварт, И.Р. Нурбахш, Д. Скарамуцца; пер. с англ. – 2–е изд. – М.: Техносфера, 2011. – 472 с.

4. Каплина М. С. Разработка алгоритма поиска выхода из лабиринта [Электронный ресурс] // НАУКА, ОБРАЗОВАНИЕ, ИННОВАЦИИ: ПУТИ РАЗВИТИЯ. – 2015. – Т. 10, № 3. – С. 135–139. – Режим доступа: http://elibrary.ru/item.asp?id=23913705 – Дата доступа: 27.05.2024.

5. Кучерский Р. В. Модели и алгоритмы картографирования среды и планирования движений автономных мобильных роботов для мониторинга лабиринтов [Электронный ресурс] – Режим доступа: http://dlib.rsl.ru/01007507937 – Дата доступа: 14.11.2024.

6. Методы определения объектов на изображении [Электронный ресурс] // Молодой ученый. – Режим доступа: https://moluch.ru/archive/193/48447/. – Дата доступа: 24.10.2024.

7. Обнаружение движущихся объектов с помощью OpenCV с использованием обнаружения контуров и вычитания фона [Электронный ресурс] // Хабр. – Режим доступа: https://habr.com/ru/articles/786436/. – Дата доступа: 12.04.2025.

8. Обход графа: поиск в глубину и поиск в ширину простыми словами на примере JavaScript [Электронный ресурс] // Хабр. – 2020. – Режим доступа: https://habr.com/ru/articles/504374/. – Дата доступа: 09.11.2024.

9. Павлов А. С. Методика планирования траектории движения группы мобильных роботов в неизвестной замкнутой среде с препятствиями //Системы управления, связи и безопасности. – 2021. – №. 3. – С. 3.

10. Роботы Amazon справляются со своими задачами в 4 раза быстрее человека [Электронный ресурс] // PCNews. – Режим доступа: http://pcnews.ru/blogs/roboty_amazon_spravlautsa_so_svoimi_zadacami_v_4_raza_bystree_celoveka-706076.html. – Дата доступа: 12.03.2025.

11. Седин А. Д., Кочетков М. А. Алгоритм управления мобильным роботом для прохождения одномаршрутного лабиринта с использованием одометрии, инерциального датчика и ультразвуковых дальномеров //Интеллектуальные системы, управление и мехатроника - 2019. – 2019. – С. 78-81.

12. Султанова А. Б. Сравнительный анализ алгоритмов поиска оптимального пути //Бюллетень науки и практики. – 2020. – Т. 6. – №. 12. – С. 248-255

13. Умаров, Х. А. Методы определения объектов на изображении / Х. А. Умаров, М. А. Умаров, Ў. Р. Хамдамов // Вестник науки. – 2019. – № 7. – С. 68–73.

14. Nadour M., Cherroun L. Using flood-fill algorithms for an autonomous mobile robot maze navigation//International Journal of System Assurance Engineering and Management. – 2022. – Т. 13. – №. 1. – С. 546-555.

15. OpenAI Gym+ROS+Gazebo: обучение автономного робота в домашних условиях. Часть 1 [Электронный ресурс] // PCNews. – Режим доступа: https://pcnews.ru/blogs/[iz_pesocnicy]_openai_gymrosgazebo_obucenie_avtonomnogo_robota_v_domasnih_usloviah_cast_1-881484.html. – Дата доступа: 04.10.2024.

16. OpenCV: Open Source Computer Vision Library [Электронный ресурс]. – Режим доступа: https://opencv.org. – Дата доступа: 01.10.2024.