

АННОТАЦИЯ

ABSTRACT

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ УПРАВЛЕНИЯ БЕСПИЛОТНЫМ АВТОМОБИЛЕМ	6
1.1 Обзор применяемого аппаратного обеспечения	6
1.2 Обзор общей структуры системы управления беспилотными автомобилями	9
1.3 Обзор методов планирования движения	14
1.3.1 Формулирование задачи планирования движения	15
1.3.2 Методы поиска на графах	16
1.3.3 Hybryd A*	18
1.3.4 State Lattice	19
1.3.5 Случайные методы	21
1.3.6 Методы интерполяции кривыми	27
1.4 Обзор некоторых систем управления беспилотными автомобилями .	28
1.4.1 Команда Junior в Darpa Urban Challenge	28
1.4.2 Команда BOSS в Darpa Urban Challenge	31
1.5 Выводы по главе	32
2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ УПРАВЛЕНИЯ ДВИЖЕНИЕМ БЕСПИЛОТНОГО АВТОМОБИЛЯ	33
2.1 Формулирование требований к разрабатываемой системе	33
2.2 Проектирование общей архитектуры системы управления движением	34
2.3 Проектирование подсистемы планирования локальной траектории .	38
2.3.1 Использование алгоритма A*	39
2.3.2 Метод интерполяции кривых	41
2.3.3 Планирование на несколько шагов	55
2.4 Выводы по главе	62
3 РЕАЛИЗАЦИЯ СИСТЕМЫ УПРАВЛЕНИЯ ДВИЖЕНИЕМ БЕСПИЛОТНОГО АВТОМОБИЛЯ	64
3.1 Использование ROS в качестве основы для системы управления . .	64

3.2 Постройка мобильной платформы	69
3.3 Реализация подсистемы распознавание препятствий	72
3.4 Реализация подсистемы планирования траектории	74
3.5 Реализация подсистемы следования траектории	76
3.6 Выводы по главе	80
4 ЭКСПЕРИМЕНТЫ И ОЦЕНКА РЕЗУЛЬТАТОВ РАБОТЫ	81
4.1 Экспериментальное исследование подсистемы движения по траектории	81
4.2 Экспериментальное исследование подсистемы планирования локального движения	81
4.3 Выводы по главе	83
ЗАКЛЮЧЕНИЕ	85

ВВЕДЕНИЕ

В настоящее время активно ведутся исследования и разработки в области беспилотных транспортных средств, в частности, беспилотных автомобилей. Беспилотные транспортные средства являются очень перспективной технологией, на которую делают ставку многие крупные автопроизводители и ИТ-компании. В частности, исследованиями и разработками в этой области занимаются такие компании как Waymo (Google), Tesla, Audi [**company_1**], Яндекс [**company_yandex**], GM, Apple, Uber, Intel совместно с BMW и ряд других. Наибольших успехов в этом достигли Waymo, которая запустила тестовый коммерческий сервис беспилотного такси для ограниченного круга испытателей в нескольких городах штата Аризона, Яндекс, запустивший беспилотное такси в Иннополисе и технопарке Сколково, и компания Tesla, выпускающая серийные автомобили с ограниченными функциями автопилота.

Массовое применение беспилотных автомобилей на дорогах общего пользования может привести к ряду положительных изменений, таких как

- уменьшение человеческого фактора, что уменьшит количество ДТП и увеличит безопасность на дорогах;
- увеличению пропускной способности дорог и уменьшение пробок, потому что беспилотные автомобили смогут поддерживать меньшее расстояние друг с другом, быстрее реагировать на изменения дорожной ситуации;
- увеличение плотности парковок и иных инфраструктурных объектов по тем же причинам, что позволит сэкономить площадь;
- уменьшение количества личных автомобилей, по причине того, что, согласно анализу, один общественный беспилотный автомобиль может заменить девять–тринацать личных автомобилей без компромиссов в текущих сценариях использования [**overview_private_ownership**], что еще больше сократит нагрузку на инфраструктуру, уменьшит пробки и сократит вредные выбросы.

Эти и ряд других возможностей делают беспилотные транспортные

средства крайне полезными и перспективными в будущем.

Поэтому актуальной является работа по исследованию и разработке системы управления беспилотными наземными транспортными средствами.

Целью данной диссертационной работы является разработка и реализация методов управление движением беспилотного автомобиля. Для достижения данной цели необходимо решить следующие задачи:

- 1) анализ существующих подходов к задаче управления беспилотными автомобилями в целом, выделение типичных подсистем, анализ методов планирования движения для наземных транспортных средств;
- 2) проектирование и реализация подсистемы формирования программной траектории;
- 3) проектирование и реализация подсистемы движения по траектории,
- 4) проведение экспериментов и оценка результатов работы.

В первой главе настоящей работы приведен обзор и анализ существующих подходов к построению систем управления беспилотными автомобилями в целом, анализ возможных структур систем управления, выделение типичных подсистем.

Во второй главе рассматривается проектирование подсистем, отвечающих за управление движением автономного автомобиля.

В третьей главе рассматривается реализация рассмотренных подсистем, а также методика их применения для управления беспилотным автомобилем.

В четвертой главе рассматриваются экспериментальная часть работы, связанная с использования разработанной системы для проведения исследования движений автомобиля.

В разделе "Выводы" подводятся итоги создания подсистемы управления беспилотным автомобилем и рассматриваются результаты экспериментов.

Научная новизна и актуальность. Хотя, актуальность вроде расписал, но не сказал явно это слово, надо сказать

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ УПРАВЛЕНИЯ БЕСПИЛОТНЫМ АВТОМОБИЛЕМ

В этой главе рассматривается обзор существующих систем управление беспилотными автомобилями. Определены основные принципы и выделены основные подсистемы системы управления. Произведен обзор способов построения траекторий движения и способов удержания этой траектории.

1.1 Обзор применяемого аппаратного обеспечения

Беспилотные автомобили должны функционировать в условиях сложной окружающей среды с большим количеством различных неподвижных и движущихся препятствий различных конфигураций, в окружении других участников дорожного движения, пешеходов и т.п. Для обеспечения высокого уровня безопасности для пассажиров и других представителей дорожного движения беспилотный автомобиль должен оперативно и адекватно реагировать на изменяющиеся условия окружения, для этого автомобиль должен быть оснащен различными датчиками и реализовывать надежные и эффективные алгоритмы обработки данных.

Важным этапом в развитии беспилотных автомобилей были соревнования Darpa Grand Challenge (2004г, 2005г) и Darpa Urban Challenge (2007г), в ходе которых команды соревновались в разработке и постройке беспилотного автомобиля, который сможет самостоятельно проехать в трассу в загородных условиях (2004г, 2005г) и в городских условиях (2007г). Некоторые команды опубликовали ряд работ, в которых описаны подходы, использованные ими при разработке беспилотного автомобиля.

Главным датчиком большинства беспилотных автомобилей является 3D LIDAR (Light Identification Detection and Ranging) — вращающийся многолучевой лазерный дальномер, позволяющий получать данные о конфигурации окружающего пространства в трехмерном пространстве на большом расстоянии. Так, например, ЛИДАР Velodyne HDL-64E способен обнаруживать объекты с высокой отражающей способностью (напри-

мер, автомобили) на расстоянии до 120 м, а объекты с низкой отражающей способностью на расстоянии до 50 метров [sensor_hdl64_manual]. ЛИДАР позволяет получить большое количество информации об окружающем пространстве и с его помощью значительно легче и надежнее реализовать такие задачи, как обнаружение препятствий, определение дороги и т.п. Из рассмотренных команд, участвовавших в Darpa Urban Challenge: Junior (Стэнфордский университет) [darpa_junior], Talos (Массачусетский технологический институт)[darpa_mit], BOSS (несколько участников из университета Карнеги-Меллона, компаний General Motors, Caterpillar, Continental, Intel) [darpa_boss], AnnieWAY (технологический институт Карлсруэ) [darpa_annieway] все команды использовали шестидесятилучевой лучевой ЛИДАР Velodyne HDL-64E в качестве основного источника информации об окружающем пространстве. Дополнительно применялись однолучевые 2D-ЛИДАРЫ в различных конфигурациях, для того, чтобы получить лучшее покрытие слепых зон, расположенные по бокам, на переднем и/или заднем бамперах, на крыше. ЛИДАРы не лишены недостатков. Главным их недостатком является высокая стоимость, что делает их не лучшим кандидатом для применения в массовых коммерческих автомобилях, так например шестнадцатилучевой ЛИДАР Velodyne VLP-16 стоит дороже ряда бюджетных автомобилей. Помимо этого, работа ЛИДАРа затруднена или во все невозможна в дождь, снег, туман.

Несмотря на большой объем и сравнительно высокую точность данных, получаемых с помощью ЛИДАРа, одного ЛИДАРа не достаточно для построения системы компьютерного зрения беспилотного автомобиля. В дополнение к ЛИДАРам, все системы комплектуются одной, но чаще несколькими, цветными и/или черно-белыми камерами, а также радарами.

Камеры, наряду с ЛИДАРом, являются важнейшими датчиками системы компьютерного зрения беспилотного автомобиля. С помощью камер производится распознавание дорожных знаков, светофоров, дорожной разметки. В настоящее время, в связи со значительным прогрессом в области искусственных нейронных сетей, можно с высокой точностью производить детектирование различных объектов [cv_nn_object2d_review], как в 2D [cv_nn_object2d_ssd2], так и в 3D [cv_nn_object3d_voxelnet2], [cv_nn_object3d_pointnet2], а так-

же сегментацию [**cv_nn_segmentation_deeplabv3**], т.е. определять, какие пиксели изображения относятся к тем и или иным к классам, таким образом можно детектировать дорогу и ровное пространство, доступное для движения. Прогресс в этой области делает возможным реализацию системы компьютерного зрения для беспилотного автомобиля без использования ЛИДАРа, что существенно снижает сложность и стоимость такой системы, что важно для серийных автомобилей. В настоящее время автомобили Tesla, оснащенные автопилотом с ограниченной функциональностью, используют только камеры. Тем мне менее, использование ЛИДАРа значительно увеличивает точность и надежность системы, и ЛИДАРы используются во всех проектах беспилотных автомобилей. Комбинируя данные с ЛИДАРа и камер (так называемая задача *sensor fusion*), можно существенно повысить точность таких задач, как распознавание и отслеживание препятствий, определение дороги.

Другим важным датчиком, применяемым в беспилотных автомобилях, является радар. За последние годы применение радаров в автотранспорте (*automotive radar*) активно развивалось. Применение радаров не ограничивается полностью беспилотными автомобилями, они находят место и в обычных автомобилях в системах помощи водителю (*advanced driver-assistance system, ADAS*) и в системах активной безопасности [**sensor_radar_overview**], [**sensor_radar_overview_2**]. Радары обладают рядом возможностей, недоступных другим датчикам, таких как возможность работы вне зависимости от погодных условий и уровня освещенности, большая (до 200м) дальность, возможность определять положение объектов и непосредственно определять их скорость с помощью эффекта Доплера.

Например, команда BOSS [**darpa_boss**] в Darpa Urban Challenge применяла радар совместно с ЛИДАРом для отслеживания объектов. Объекты отслеживались независимо, используя особенности датчиков (непосредственное измерение скорости объектов с помощью радара играло важную роль в алгоритме отслеживания) и затем результаты объединялись.

В таблице 1 приведено сравнение основных свойств и областей применения рассмотренных выше датчиков.

Таблица 1 – Сравнение применяемых датчиков

Датчик	Получаемая информация	Дальность	Влияние освещенности	Влияние погодных условий
Камера	Цветное или ч/б изображение	Не позволяет определить расстояние	Да	Да
Стереокамера	Облако точек	до 20 м	Да	Да
3D ЛИДАР	Облако точек 360°	до 120 м	Малое	Да
2D ЛИДАР	Положение препятствий (2D)	до 20 м [jetson — car — zed]	Малое	Да
Радар	Положение и скорость препятствий (2D)	до 200 м	Нет	Нет

1.2 Обзор общей структуры системы управления беспилотными автомобилями

Несмотря на то, что было представлено большое количество различных систем управления беспилотными автомобилями, все они обладают схожей структурой. В общем виде, систему управления беспилотным автомобилем можно разделить на следующие подсистемы:

- интерфейс сенсоров, позволяющий получать данные от сенсоров;
- подсистема восприятия (perception), осуществляющая построение комплексной информации об окружающем пространстве, на основе данных от сенсоров, определяя положение автомобиля, дорогу, статические и динамические препятствия, распознавания дорожные знаки, светофоры, разметку, и формирующая в конечном итоге цифровую карту окружающего пространства, которая в дальнейшем используется другими подсистемами;
- подсистема управления движением, осуществляющая принятие решений и построение безопасной и достижимой траектории и осуществляющая движение по траектории, формирования управляющих сигналов, таких как угол поворота руля, газ, тормоз.

Подобная высокоуровневая архитектура применена как командами Darpa Urban Challenge [[darpa_junior](#)], [[darpa_mit](#)], [[darpa_boss](#)], [[darpa_annieway](#)], так и описана в более современных статьях, таких как [[developing_car_1](#)], [[car_proud](#)], [[car_race](#)].

Архитектура обобщенной системы управления автономным автомобилем, построенная на основе анализа ряда работ [[motion_planning_review](#)], представлена на рисунке 1.

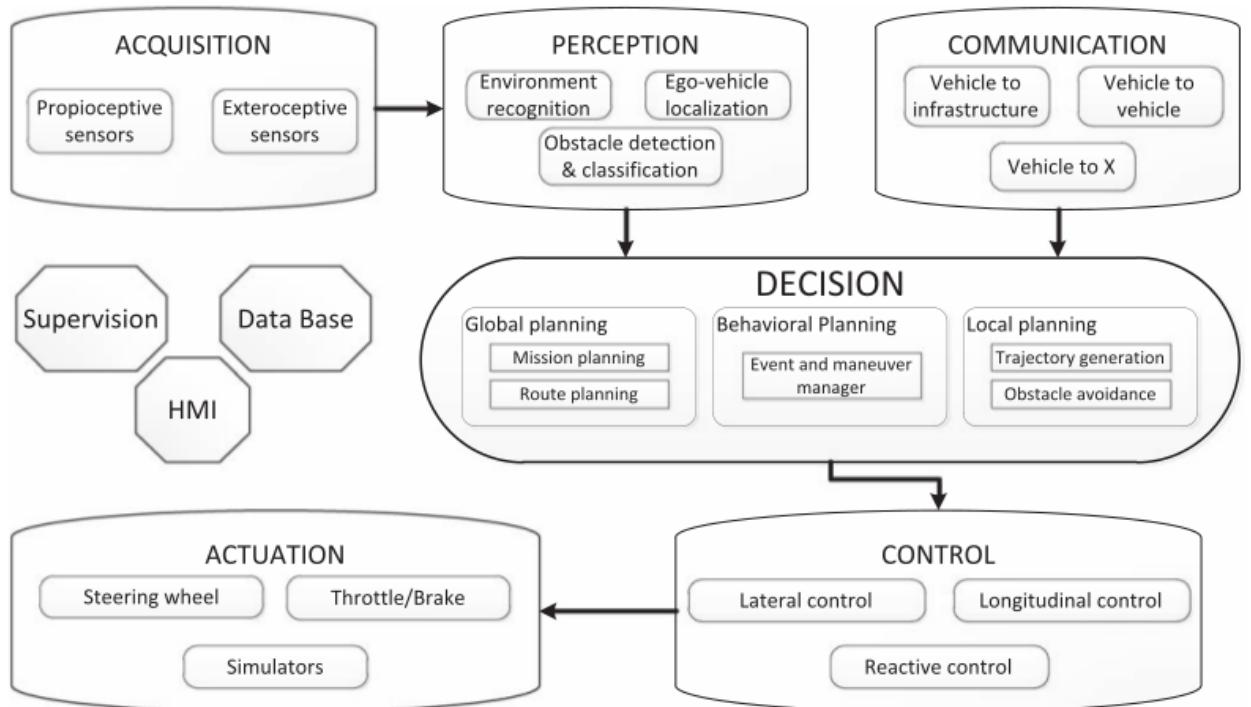


Рисунок 1 – Обобщенная абстракция архитектуры управления автономными автомобилями [[motion_planning_review](#)]

Подсистема восприятия обычно состоит из следующих компонентов, состав и взаимодействие которых, в прочем, могут отличаться в зависимости от назначения и уровня автономности беспилотного автомобиля:

- подсистема локализации, осуществляющая определения положения автомобиля с помощью GPS и системы одновременной картографии и навигации (SLAM),
- подсистема выделения дороги (ground detection/road detection), определяющая, какие области пространства вокруг доступны для передвижения,
- подсистема детектирования (object detection) и отслеживания (object tracking) объектов, позволяющая определять положение дина-

мических препятствий вокруг автомобиля и их скорость и направление движения,

- подсистемы распознавания дорожных знаков, светофоров, дорожной разметки и т.п.

Процесс планирования движения и принятия решений в современных беспилотных автомобилях обычно представлен в виде иерархии: планирование маршрута (route planning), принятие решений (behaviour planning, decision making), локально планирование (local motion planning) и управление с обратной связью [**motion_planning_review_2**]. Точное разделение на уровни обычно размыто и может отличаться в различных работах, но общая структура сохраняется. Пример того, как разделяются задачи по уровням иерархии представлен на рисунке 2.

На верхнем уровне осуществляется планирование маршрута по дорожной сети. Затем следуют уровень планирования поведения, который принимает решения и формирует локальные навигационные задачи, которые приближают автомобиль к выполнению высокоуровневой задачи и удовлетворяют правилам дорожного движения. Затем локальный планировщик формирует непрерывный путь в окружающем пространстве, который выполняет локальную навигационную задачу. Система управления с обратной связью осуществляет выполнение запланированного движения и коррекцию ошибок.

Система планирования маршрута должна выбирать как можно более оптимальный путь по дорожной сети от текущего положения к точке назначения. Типичным решением является представление дорожной сети как ориентированного графа, ребрам которого назначаются веса, соответствующие стоимости движения по данному сегменту дороги (в простейшем случае — расстояние). Тогда задача построения маршрутка может быть сформулирована как задача нахождения пути на графике с наименьшей стоимостью. Участники Darpa Urban Challenge решали эту задачу самостоятельно, используя хорошо известные алгоритмы, такие как алгоритмы Дейкстры, A* или их модификации. Это было возможно, потому что область проведения Darpa Urban Challenge была небольшая. В реальных сценариях применение простейших алгоритмов невозможно, потому что дорожная сеть может содержать миллионы узлов. Но это и не требуется, потому что в настоящее время в настоящее время существует большое количество картографических сервисов.

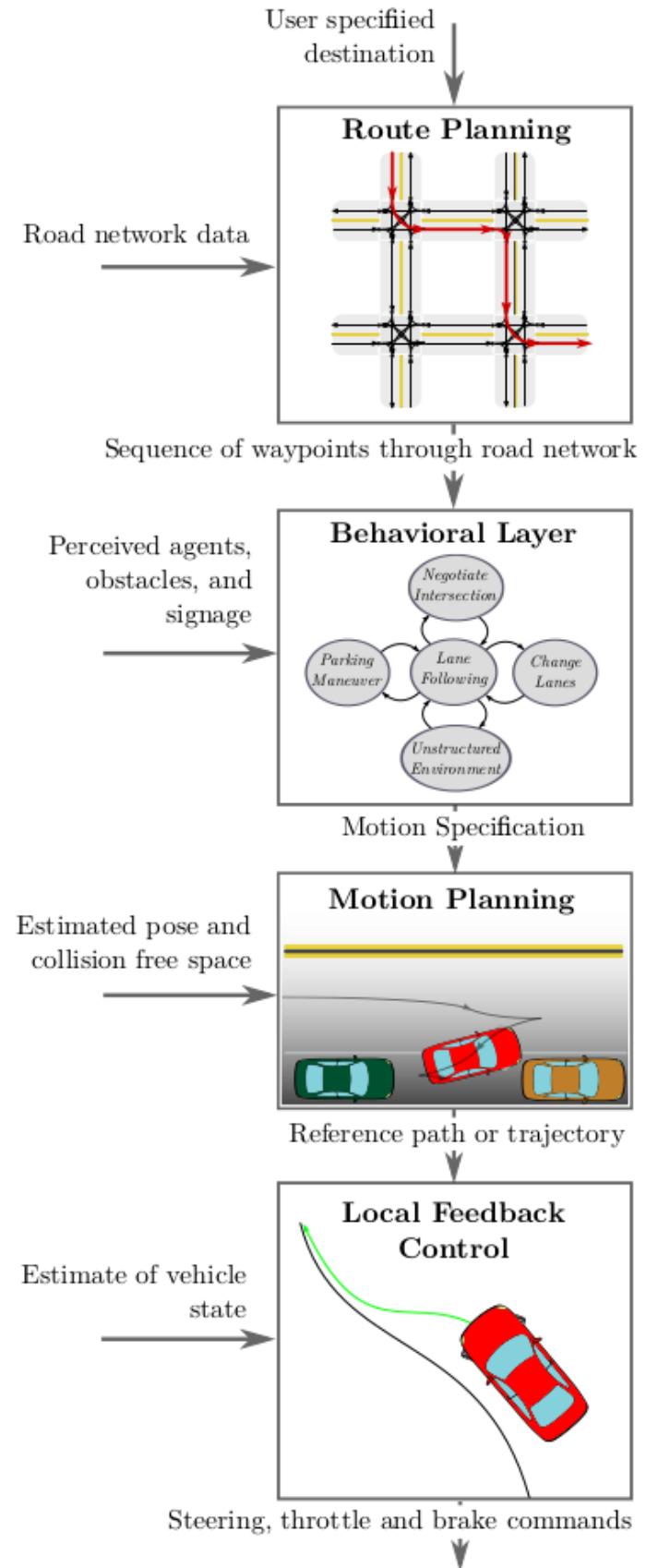


Рисунок 2 – Иерархия системы управления

сов, реализующих эффективные алгоритмы построения маршрутов вплоть до континентального масштаба и предоставляющих подобную функциональность в виде API. Поэтому в данной работе не рассматривается глобальное планирование маршрута, так как реализовывать его самостоятельно не имеет смысла.

После того, как глобальный маршрут был построен, автономный автомобиль должен уметь двигаться по этому маршруту и взаимодействовать с другими участниками дорожного движения в соответствии с правилами дорожного движения. Планировщик поведения отвечает за выбор подходящего поведения в каждый момент времени, основываясь на наблюдаемой дорожной ситуации. Например, когда автомобиль достигает стоп-линии, планировщик поведения формирует команду остановки. Правила дорожного движения определяют правильно поведения в различных ситуациях. Варианты поведения и возможные дорожные ситуации представляют собой конечные множества, поэтому естественным будет представление планировщика движения в виде конечного автомата. Большинство участников DARPA Urban Challenge применяли конечные автоматы для задачи принятия решений.

Основной проблемой является неопределенность намерений остальных участников дорожного движения. Проблема предсказания намерений участников движения активно изучается и существуют различные подходы, от простой линейно экстраполяции [**darpa_junior_frenet**] движения до предсказания движения с помощью Deep Learning [**nn_motion_prediction**], что является наиболее перспективным способом в настоящее время. Неопределенность поведения участников дорожного движения часто учитывается в планировании поведения спомощью вероятностного формализма, например, применяя марковский процесс принятия решений.

После того, как планировщик поведения определил поведение, которое должно выполняться в данной дорожной ситуации, например, удерживать полосу, выполнит обгон или остановиться на светофоре, выбранное поведение должно быть представлено в виде пути или траектории, которые, в свою очередь, могут быть выполнены низкоуровневым регулятором с обратной связью. Эта траектория должна избегать препятствий, быть достижимой для автомобиля согласно его кинематике и динамике и, наконец, быть комфортной для пассажиров. Эта задача соответствует стандартной задаче

планирования движения (motion planning) в робототехнике.

И, наконец, регулятор с обратной связью отвечает за то, чтобы сформировать управляющие сигналы, такие как поворот руля, газ и торможение, чтобы точно следовать программной траектории.

1.3 Обзор методов планирования движения

Локальный планировщик движения отвечает за формирование безопасной, достижимой и комфортной для пассажиров траектории из текущего положение автомобиля до следующей цели, определенной планировщиком поведения. В зависимости от ситуации, это могут быть различные цели, такие как удержание полосы, следование на заданном расстоянии от впереди идущего автомобиля, остановка и др. Планировщик движения учитывает информацию о статических и динамических препятствиях, чтобы сгенерировать траекторию, которая будет избегать препятствий и будет выполнима с точки зрения кинематики и/или динамики автомобиля. В некоторых подходах рассматривается формирование оптимальной по отношению к некой целевой функции траектории.

Получаемое программное движение может представлено в виде пути или траектории. В случае планирования пути, результатом будет путь в конфигурационном пространстве автомобиля,

Задача планирования движения (motion planning) в робототехнике — процесс разбиения требуемого движения на дискретные действия, которые удовлетворяют ограничениям. Планирование движения является важной задачей робототехники и активно исследуется и разрабатывается в течении последних десятилетий. Разработано большое количество алгоритмов планирования движения, как для решения задачи в общем случае, так и для конкретных приложений. В этом разделе дается обзор методов, позволяющих осуществить задачу локального планирования беспилотного движения автономного автомобиля.

Существует большое количество методов, позволяющих решить данную задачу, такие как методы поиска на графах, случайные методы поиска (sample-based motion planning), методы интерполяции движения кривыми, методы численной оптимизации и другие. Обзор методов планирования

движения в приложении к беспилотным автомобилям рассмотрен в работах [motion_planning_review], [motion_planning_overview_obstacles], [motion_planning_overview_modern], [motion_planning_review_2], [motion_planning_review_3].

Различные способы планирования движения имеют свои преимущества и недостатки. Например, одни методы позволяют более полно решать задачу планирования движения, но являются более вычислительно сложными, что затрудняет их применение в реальном времени на дорогах с быстро меняющейся ситуацией. Чтобы решить эту проблему, многие команды в DARPA Urban Challenge не ограничивались одним способом планирования движения. Распространенным подходом было применять один планировщик движения при движении по дорогам, а другой - при движении в неструктурированном окружении, например, при задаче парковки.

Задача планирования движения в применении к автомобилю в общем виде является нетривиальной, потому что динамика автомобиля описывается сложными дифференциальными уравнениями, имеющими, что немаловажно, неголономные связи [car_frund]правиль-но написать. Многие исследователи применяют различные упрощенные модели динамики автомобиля [car_dynamics_1]. Очень распространенной является "велосипедная" модель динамики автомобиля, которая применяется в ряде работ [car_dynamics_bycicle_model_1], [car_dynamics_bycicle_model_2], [darpa_annieway_navigation].

1.3.1 Формулирование задачи планирования движения

Перед рассмотрением алгоритмов планирования движения необходимо определить несколько понятий, которые применяются в теории автоматизированного управления и сформулировать задачу планирования движения.

Конфигурационное пространство \mathcal{C} системы — это совокупность всех ее обобщенных координат.

Дано:

- пространство конфигураций \mathcal{C} робота;
- множество конфигураций, соответствующее препятствиям $\mathcal{C}_o \in \mathcal{C}$, соответственно множество свободных конфигураций $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_o$;

- начальные и конечные конфигурации q_s , q_g .

Задача планирования движения может быть сформулирована как планирование пути или планирование траектории. В первом случае, требуется найти функцию $\tau(\alpha) : [0, 1] \rightarrow \mathcal{C}_{free}$, представляющую путь робота в конфигурационном пространстве, где $\tau(0) = q_s$, $\tau(1) = q_g$. В этом случае, решение не определяет, как этот пройден автомобилем. Планировщик движения может выбрать профиль скорости для этого пути, как это сделано в **[darpa_boss]**, или делегировать эту задачу нижележащим подсистемам.

В случае планирования траектории, явно учитывается время движения. В таком случае требуется найти функцию $\pi(t) : [0, T] \rightarrow C$, где T - горизонт планирования. В отличие от планирования пути, планирование траектории явно указывает, как должна меняться конфигурация с течением времени.

Требуется это более внятно и корректно сформулировать.

1.3.2 Методы поиска на графах

Для решения задачи планирования движения можно применять алгоритмы поиска на графах, такие как алгоритм Дейкстры, алгоритм A* или их модификации. Для применения алгоритмов поиска на графах к задаче планирования движения необходимо сначала представить конфигурационное пространство в виде графа.

В данном методе большую роль играет способ построения графа, а именно выбор опорных точек и ребер, которые их соединяют.

Простейшими, но крайне широко распространенными вариантами представления конфигурационного пространства в виде графа являются методы клеточной декомпозиции. Наиболее распространенным вариантом является приближенная клеточная декомпозиция, когда пространство разбивается с помощью равномерной сетки (grid map) **[motion_planning_overview_obstacles]**. Преимуществом этого метода является то, что его легко реализовать и легко представить конфигурацию окружающего пространства в виде сетки. Этот метод широко применяется в ряде существующих реализаций поиска пути, таких как navigation stack в ROS **[ros_navigation]**. Недостатком этого метода является существенное

увеличение размера графа при уменьшении шага дискретизации, увеличении области, в которой осуществляется планирование и, особенно, при увеличении размерности конфигурационного пространства. Подобные методы в основном применяются для геометрического планирования на плоскости, с увеличением числа степеней свободы (повороты, трехмерное пространство, трехмерное пространство с поворотами) количество вершин в графе становится слишком большим для эффективного применения данного метода.

Другим способом выбора опорных точек графа является использование диаграммы видимости. [**motion_planning_overview_obstacles**] Диаграмма видимости определяется как неориентированный граф, вершины которого включают множество вершин препятствий, а ребра соединяют некоторые вершины таким образом, что никакое ребро не пересекается с препятствиями. Для использования диаграммы видимости необходимо, чтобы препятствия имели форму многоугольника или многогранника. Поскольку путь строится по вершинам препятствий, и часть пути совпадает с краями препятствий, существует определенная опасность столкновения с препятствиями. Кроме того, при увеличении количества препятствий возрастает сложность графа, причем этот рост очень быстрый. Подобный подход редко применяется для беспилотных автомобилей.

Еще одним методом построение графа, который сильно распространен в задачах планирования [**darpa_annieway_navigation**], [**motion_planning_overview_obstacles**], являются диаграммы Вороного. Диаграмма Вороного конечного множества точек на плоскости представляет собой разбиение плоскости таким образом, что каждая область образует геометрическое место точек (локус), каждая из которых более близка к одному элементу множества, чем к другому. Преимуществом диаграмм Вороного для планирования движения является то, что с их помощью можно построить путь, наиболее удаленный от всех препятствий, и, следовательно, наиболее безопасный.

Команда "BOSS"Darpa Urban Challenge применяла метод поиска на графах для планирования движения [**darpa_boss**], [**darpa_boss_2**], [**darpa_boss_3**]. Для графа они применяют т.н. решетку состояний (state lattice). Узлы графа представляют собой дискретизированные состояния в пространстве конфигураций, а каждое ребро графа представляет достижи-

мый путь. В отличие от более распространенных подходов, применяющих сетки, клетки которых просто имеют 4 или 8 соединенных соседей, в этом подходе гарантируется, что полученный путь будет достижимым.

Наиболее применительны методы поиска на графах для планирования движения в неструктурированном окружении.

Далее будут рассмотрены некоторые модификации известных алгоритмов поиска пути на графах, ориентированные на нахождение кинематически и динамически достижимых путей.

1.3.3 Hybrid A*

Из-за нелинейной динамики автомобиля, путь, построенный с помощью поиска на регулярной прямоугольной дискретной сетке будет кинематически и динамически недостижим для автомобиля. Было предложено много вариантов решения этой проблемы, так например, алгоритм Hybrid A* [**motion_planning_hybrid_a_star**], предложенный командой "Junior" DUC [**darpa_junior**]. В отличие от алгоритма A*, примененного к регулярной сетке, имеющей 4 или 8 дискретных соединений между соседними ячейками, Hybrid A* формирует набор траекторий, учитывающих нелинейную динамику и непрерывную природу движения автомобиля (рисунок 3).

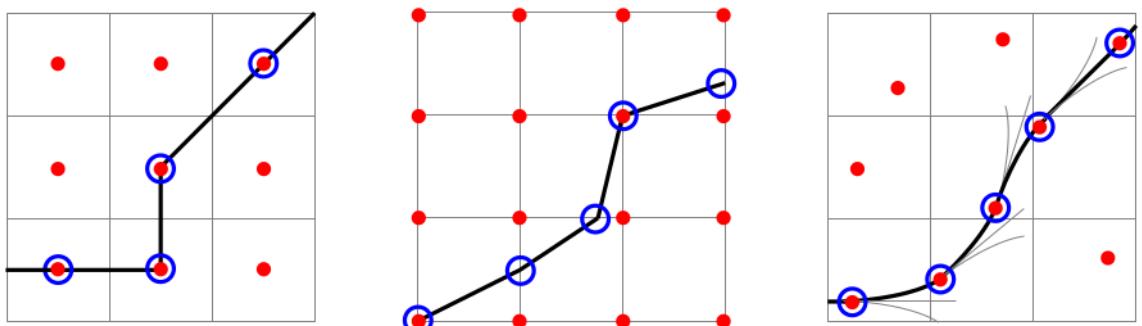


Рисунок 3 – Hybrid A* в сравнении с A* [**darpa_junior**]

Алгоритм работает с дискретными сетками в трехмерном пространстве состояний x, y, θ , где x, y - положение автомобиля, а θ - ориентация. Рассмотрим работу алгоритма. Допустим, что в некий момент времени состояние автомобиля $< x, y, \theta >$ и оно соответствует ячейке c_i в дискретизированном пространстве состояний. Сопоставим с этой ячейкой непрерывные координаты: $x_i = x, y_i = y, \theta_i = \theta$. Далее решаем задачу прямой динамики,

интегрируя движение автомобиля при подаче управления u в течении заданного времени t . Конечным состоянием будет $\langle x', y', \theta' \rangle$, попадающее в дискретную клетку c_j . Аналогичным образом сопоставим непрерывные координаты с этой клеткой: $x_j = x'$, $y_j = y'$, $\theta_j = \theta'$. Это позволяет гарантировать, что мы имеем непрерывную траекторию из клетки c_i и c_j , для которой гарантируется кинамическая и динамическая достижимость (разумеется, при использовании достаточно точной динамической модели), а также соответствующее управление u , что не гарантируется для обычного A* алгоритма. Дальнейший поиск по этому графу осуществляется аналогично обычному алгоритму A*.

1.3.4 State Lattice

Другой широко распространенной модификацией графовых алгоритмов является семейство алгоритмов, называемых в англоязычной литературе "State Lattice" (решетка состояний). Алгоритм был предложен в работе [motion_planning_lattice_1] и близок по своей идеи к алгоритму Hybrid A*, рассмотренному ранее. В работе представлено пространство поиска, называемое решеткой состояний, позволяющее представить планирование неголономного движения как задачу поиска на графике.

Решетка состояний представляет собой дискретное множество всех достижимых конфигураций системы. Ее построение осуществляется путем дискретизации пространства состояний в виде многомерной решетки, состоящих из достижимых состояний, и попытки соединить начальное состояние со всеми узлами решетки с помощью достижимых путей в качестве ребер. В общем случае, решетка состояний содержит все достижимые пути (с заданной дискретизацией), что подразумевает, что если автомобиль можете переместиться из одного состояния в другое, то решетка состояний содержит последовательность путей для осуществления этого маневра.

Подобно обычной сетке, решетка состояний преобразует задачу планирования в непрерывном пространстве в задачу формирования последовательности выбора из нескольких дискретных альтернативных состояний, но в отличие от обычной сетки, решетка состояний строится таким образом, что ее связи являются достижимыми путями.

Построение решетки состояний требует решение обратной задачи для нахождения достижимого пути между двумя заданными состояниями. Существует ряд методов, позволяющих решить эту задачу. Несмотря на то, что метод решетки состояний по-прежнему требует решения обратной задачи, что, как было замечено выше, является нетривиальной задачей, этот метод существенно упрощает задачу планирования движения, потому что позволяет осуществлять планирование движения в виде графа в условиях наличия различных препятствий, позволяя формировать сложную траекторию, что существенно более затруднено при применении чисто математических методов решения этой задачи. В данном случае требуется решать обратную задачу лишь для некого ограниченного множества возможных комбинаций начальных и конечных состояний.

Другим немаловажным свойством этого метода является то, что в силу использования регулярной сетки, можно выделить набор состояний, инвариантный к пространственному перемещению (рисунок 4). Это позволяет заранее расчитать ограниченный набор состояний и соответствующих траекторий и управлений с помощью точных моделей и ресурсозатратных алгоритмов, а в процессе планирования движения в реальном времени пользоваться этой заранее расчитанной таблицей. Этот подход широко применяется в ряде разработок, например, в автомобиле "BOSS" DUC [**darpa_boss**].

Важным понятием в этом методе является понятие эквивалентности путей. Путь τ_1 считается эквивалентным пути τ_2 , если τ_1 содержится в определенном регионе Q вокруг τ_2 (рисунок 5). Все пути, являющиеся эквивалентными, представляются одним путем.

Для достижения эффективности, как было сказано выше, возможно дублирование некого базового набора состояний и траекторий для разных начальных состояний, таким образом позволяя возможность эффективно конструировать граф необходимого размера в реальном времени. Для реализации такой возможности во всем множестве путей, которое представляет собой решетка состояний, необходимо выделить базовое множество. Например, для обычной регулярной сетки, каждая ячейка которой соединена с четырьмя соседними, базовым множеством путей будет движение на одну клетку вперед, назад, влево и вправо. Этого простого набор путей достаточно, чтобы описать любые перемещения в этом пространстве состояний. Для решетки со-

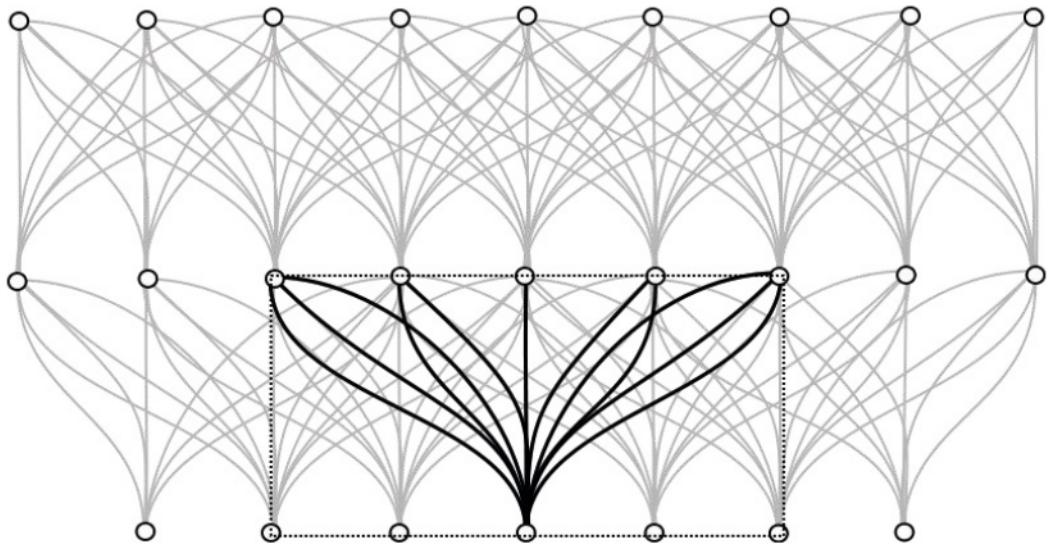


Рисунок 4 – Пример регулярной решетки состояний. Базовое множество управлений и результирующих путей (чёрный) не зависит расположения и может быть продублирован. Это позволяет очень эффективно представлять множество траекторий и решать проблему планирования движения
[motion_planning_lattice_2]

стояний применяется подобная идея. Любой путь через решетку может быть декомпозирован в последовательность путей из некого минимального набора таким образом, что эта последовательность будет эквивалентной исходному пути.

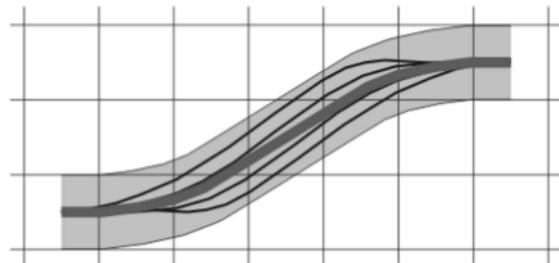


Рисунок 5 – Эквивалентность путей. Множество различных путей (чёрные линии), которые содержаться в пределах заданного региона (серая область), считаются эквивалентными

1.3.5 Случайные методы

При решении задач планирования движения в пространствах высоких размерностей, которые часто возникают в современной робототехнике (например, планирование движения твердого тела в трехмерном простран-

стве с шестью степенями свободы или планирование движения многозвенного манипулятора), классические методы, такие как методы поиска на графах, обладают очень большой вычислительной сложностью. Для решения таких задач перспективными являются случайные (sample-based) методы, такие как Probabilistic roadmap, Rapidly-exploring random graph (RRG), Rapidly-exploring random tree (RRT) [**motion_planning_rrt**], Optimal Rapidly-exploring random tree (RRT*)[**motion_planning_rrt_star**].

Идея метода заключается в том, что начиная из начального состояния строится дерево, которое может достичь конечного состояния, путем выбора случайных точек в конфигурационном пространстве. Алгоритм RRT представлен в листинге 1, а иллюстрация метода *EXTEND* на рисунке 6.

Algorithm 1 Rapidly-exploring random tree

```

function RRT( $x_{init}$ )
     $\tau.init(x_{init})$ 
    for  $k = 1$  to  $K$  do
         $x_{rand} \leftarrow RANDOM\_STATE()$ 
        EXTEND( $\tau$ ,  $x_{rand}$ )
    end for
end function
function EXTEND( $\tau$ ,  $x$ )
     $x_{near} \leftarrow NEAREST\_NEIGHBOUR(x, \tau)$ 
    if NOT_COLLIDED( $x, x_{near}, x_{new}$ ) then
         $\tau.add\_vertex(x_{new})$ 
         $\tau.add\_edge(x_{near}, x_{new}, u_{new})$ 
        if  $x_{new} = x$  then
            return REACHED
        else
            return ADVANCED
        end if
    end if
    return TRAPPED
end function

```

Особенность этого семейства алгоритмов, как и многих других итерационных методов, является их сходимость в пределе, т.е. решение будет получено при количестве итераций стремящемся к бесконечности. Тем не менее, практическое использование этого семейства алгоритмов позволяет быстро находить решение для высоких размерностей. Это свойство является преимуществом для систем реального времени, которым является беспилотный

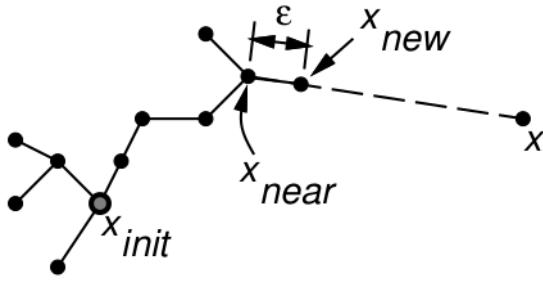


Рисунок 6 – Пример работы метода EXTEND алгоритма RRT

автомобиль. Для систем реального времени ключевой является способность выполнять работу в строго определенный интервал времени. К примеру, под планирование локальной траектории вперед на заданный горизонт планирования отводится 0.5 секунды, и превышение этого времени может привести к нарушению функционирования беспилотного автомобиля, что может создать опасную ситуацию. В случае с RRT алгоритмами, указывается максимальное время работы, после которого итерационный процесс прекращается. В этом случае, полное решение может быть не найдено, но планирование может быть повторено через некоторое время, чтобы построить недостающий участок пути.

Для применения этого алгоритма достаточно определить процедуру проверки на пересечение с препятствием *NOT_COLLIDED*.

Существует большое количество модификаций алгоритма RRT, особенно необходимо отметить алгоритм RRT* [**motion_planning_rrt_star**], являющийся асимптотически-оптимальной версией алгоритма RRT. Он основан на алгоритме rapidly-exploring random graph (RRG) и имеет отличающуюся процедуру добавления новой точки в граф. В отличие от алгоритма RRT, который прекращает свою работу, когда решение было найдено, алгоритм RRT* продолжает работу, постепенно находя все более оптимальные решения.

Отдельно стоит выделить направление под названием кинодинамическое планирование (*kinodynamic planning*), которое является актуальным в последнее время. В этом случае планирование осуществляется не в конфигурационном пространстве, а в пространстве состояний (фазовом пространстве) [**motion_planning_kinodynamic_rrt**], которое определяется следующим образом. Пусть \mathcal{C} – конфигурационное пространство, каждая конфигура-

ция $q \in \mathcal{C}$ представляет положение робота в пространстве. Обозначим пространство состояний как \mathcal{X} , в котором состояние $x \in \mathcal{X}$ определяется как $x = (q, \dot{q})$. (Состояние может определяться и с использованием производных высших порядков).

Это позволяет осуществлять планирование не только с учетом кинематической модели автомобиля, но и с учетом динамической модели с дифференциальными ограничениями и неголономными связями. Таким образом, результирующие пути будут достижимы с точки зрения динамики автомобиля, что не всегда возможно для других методов планирования, что приводит к необходимости вводить различные искусственные ограничения. Результатом такого метода планирования может быть не только траектория в пространстве состояний, которая подается на вход низкоуровневого регулятора с обратной связью, но и непосредственно последовательность управляющих сигналов, которые приведут систему к движению с такой траекторией.

Для реализации этих алгоритмов необходимо определить функции *PROPAGATE* и *STEER*. Функция *PROPAGATE* (1) формирует следующее состояние системы после применения к ней заданного управляющего воздействия, например, путем численного интегрирования.

$$x_{i+1} = \text{PROPAGATE}(x_i, u_i, \Delta t) \quad (1)$$

где x_i — текущее состояние,
 u_i — управление,
 Δt — время интегрирования.

Функция *STEER* (2) значительно сложнее. Согласно алгоритму RRT необходимо осуществлять шаг из некого состояния по направлению к случайно выбранному состоянию, то в случае с кинодинамическим планированием это невозможно сделать напрямую, так как для перехода между состояниями используется функция *PROPAGATE*, реализующая динамическую модель. Для этого потребуется определить, какие управляющие сигналы могут привести в требуемое состояние, т.е. решить обратную задачу динамики. В общем случае это невозможно. В зависимости от задачи и модели, может выбираться различная функция *STEER*, реализующая обратную задачу с той или иной точностью. Благодаря итерационному алгоритму, решение может

быть найдено даже с использованием приближенной функции. В простейшем случае, в качестве функции *STEER*, может использоваться сэмплирование случайного управляющего вектора. В этом случае, сходимость алгоритма существенно ухудшается и потребуется очень много итераций, чтобы прийти к решению.

$$u^* = \text{STEER}(x_{near}, x, \Delta t) \quad (2)$$

- где x — случайно выбранное состояние,
- x_{near} — состояние, близкое к случайному,
- u^* — примерное управление,
- Δt — время интегрирования.

Интересное решение этой проблемы было предложено командой МИТ Darpa Urban Challenge [**darpa_mit**], [**darpa_mit_2**], [**darpa_mit_3**], [**darpa_mit_3**]. Они применяли RRT для планирования системы модель автомобиль-регулятор с обратной связью, т.е. использовали модель управления по замкнутому контуру для планирования движений, в то время как в большинстве решений применяется управление по открытому контуру, т.е. учитывается только модель автомобиля.

Таким образом, sample-based методы планирования движения являются очень перспективными и способными к решению сложных задач планирования движения, где не справляются другие методы. Тем не менее, эти методы не лишены ряда существенных недостатков, главным из которых является высокая вычислительная сложность. В зависимости от сложности модели, размерности пространства, конфигурации препятствий, могут потребоваться десятки или сотни тысяч итераций, чтобы найти решение, что может не укладываться в жесткие временные рамки для системы управления беспилотным автомобилем. При типичном применении кинодинамического RRT, происходит выбор (сэмплирование) точек в пространстве управления автомобиля, т.е. входов для органов управления автомобилем. В этом подходе применяется сэмплирования входов для регулятора с обратной связью. Алгоритм RRT формирует дерево со сравнительно низкой плотностью сэмплированных точек, затем происходит моделирование системы автомобиль-регулятор, формируя реальную траекторию автомобиля (рисунок 7). Этот подход позволяет

уменьшить количество вершин дерева, которое приходится открывать алгоритму RRT, тем самым уменьшив вычислительную сложность.

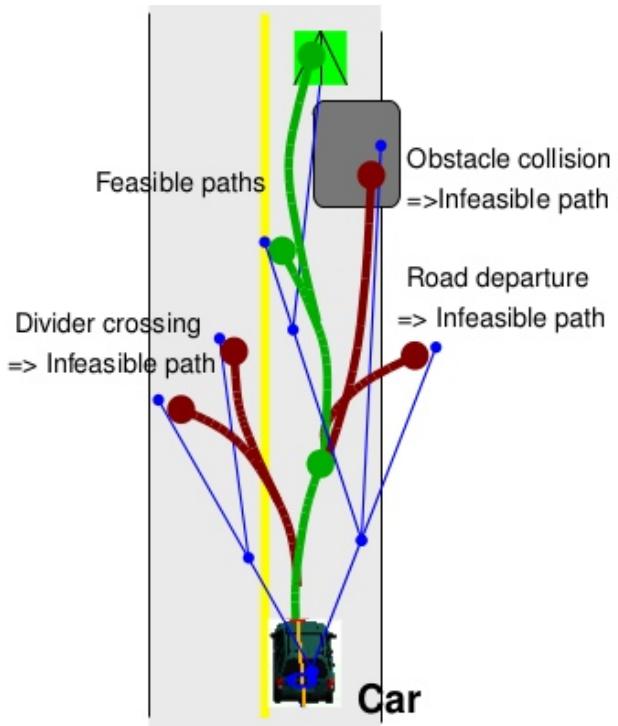


Рисунок 7 – Планирование движения с помощью RRT и регулятора с обратной связью

Существует Open-Source библиотека The Open Motion Planning Library [[motion_planning_ompl](#)], которая реализует sample-based алгоритмы поиска, таких как RRT, RRT* и ряд других, основанных на них, как для планирования в пространстве конфигураций, так и для планирования в пространстве состояний (kinodynamic planning). Библиотека легко интегрируется с любой задачей планирования движения, поскольку не имеет никаких ограничений на тип задачи, способы представления препятствий, используемые динамические модели. Программисту достаточно определить вышеназванные функции *NOT_COLLIDED*, *PROPAGATE*, *STEER* в соответствии с используемым способом представления препятствий и моделью робота. Библиотека имеет ряд готовых реализаций для широко распространенных пространств конфигурации, состояний и управления, таких как $SO(2)$, $SO(3)$, $SE(2)$, $SE(3)$, но возможно определить и свои.

1.3.6 Методы интерполяции кривыми

Следующей группой методов, применяемых для реализации задачи планирования движения беспилотного автомобиля, является интерполяция траектории движения с помощью каких-либо кривых или иных линий.

Эти алгоритмы принимают набор точек (например, данный набор точек пути, описывающих глобальную дорожную карту), генерируя новый набор данных (более плавный путь) удовлетворяющий требования на непрерывность траектории, ограничения транспортного средства и учитывая динамическую среду, в которой движется автомобиль.

В отличие от методов, рассмотренных ранее, которые являются универсальными и могут применяться к решению различных задач планирования движения, применительно к беспилотному автомобилю, это задача движения в неструктурированном окружении, эти методы широко применяются для построения траектории при движении по дороге.

Планировщики интерполяционных кривых реализуют различные методы сглаживания траектории и генерации кривых. Могут применяться различные кривые для интерполяции, такие как линии и окружности, клотоиды, кривые Безье или полиномы. Полиномы зачастую применяются для того, чтобы построить участок траектории, удовлетворяющий граничным условиям (например, положению и скорости).

Так в работах [**darpa_junior_path_planning_1**], [**darpa_junior_frenet**], опубликованных по результатам участия команды Стэнфордского университета в Darpa Urban Challenge, рассматривается планирование траектории с помощью полиномов пятого порядка в подвижной системе координат, связанной с желаемой траекторией, основанной на трехграннике Френе. В этой работе осуществляется независимое формирование продольной и поперечных траекторий, обе представлены в форме полиномов пятого порядка. Алгоритм генерирует набор кривых путем варьирования конечных условий, а затем осуществляется выбор оптимальной траектории с использованием различных функций стоимости.

Стоит отметить, что формирование большого набора траекторий с последующим выбором оптимальной является распространенным подходов в решении задачи планирования движения.

В работе [motion_planning_interpolate_path_optimization] представлен похожий подход, отличающийся тем, что после нахождения оптимальной траектории из набора дискретных траекторий, осуществляется дополнительная процедура оптимизации параметров полиномов с целью получить более оптимальную траекторию.

В работе [motion_planning_interpolate_bezier] применяются кривые Безье пятого порядка для формирования траектории движения автомобиля с моделью Аккермана. Алгоритм получает на вход набор путевых точек и формирует путь, проходящий через них с помощью набора кривых Безье. Каждый сегмент определяется положением точки, первой и второй производной, т.е. скоростью и ускорением автомобиля. Полученная кривая имеет гладкие стыки между сегментами. Затем осуществляется оптимизация кривой путем изменения положений и первых производных в опорных точках, чтобы оптимизировать длину пути.

1.4 Обзор некоторых систем управления беспилотными автомобилями

В этом разделе будут более детально рассмотрены некоторые решения в области планирования движения беспилотных автомобилей, в основном, на основе публикаций участников Darpa Urban Challenge.

1.4.1 Команда Junior в Darpa Urban Challenge

В работах [darpa_junior_path_planning_1], [darpa_junior_frenet], опубликованных по результатам участия команды Стэнфордского университета в Darpa Urban Challenge, рассматривается планирование траектории с помощью полиномов пятого порядка в подвижной системе координат, связанной с желаемой траекторией, основанной на трехграннике Френе. Иллюстрация этого способа представлена на рисунке 8, где $s(t)$ — длина дуги, пройденная автомобилем, \vec{t}_r — направляющий вектор системы координат, \vec{n}_r — нормальный вектор системы координат, $\vec{x}(s, d)$, \vec{n}_x , \vec{t}_x — положение и направляющие векторы локальной системы координат автомобиля соответственно и $d(t)$ — отклонение автомобиля от желаемой траектории.

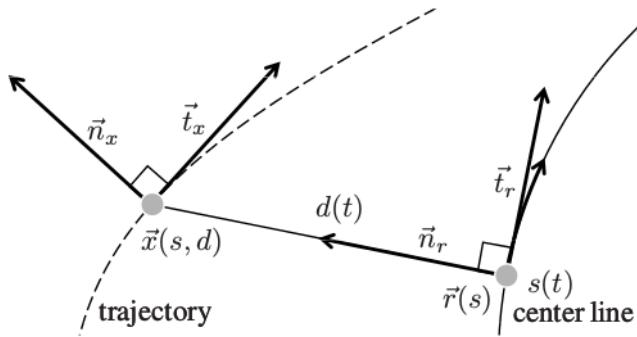


Рисунок 8 – Система координат для планирование траектории

Планирование осуществляется независимо для продольного и поперечного движения, а затем полученные полиномы объединяются в траекторию и переводятся в декартову систему координат. Выбирается оптимальная траектория, минимизирующая функцию стоимости. За основу функции стоимости взят интеграл третьей производной, рывка (jerk):

$$J = \int_{t_0}^{t_1} \ddot{p}^2 d\tau \quad (3)$$

где p — положение (продольное или поперечное),

t_0, t_1 — время совершения маневра.

Полиномы пятого порядка выбраны по причине того, что оптимальным решением для минимизации этого интеграла.

Так как при планировании движения требуется учитывать препятствия, задача оптимизации для нахождения коэффициентов полинома, минимизирующего стоимость крайне затруднительно. Вместо этого в работе предложен другой подход, являющийся типичным для подомных методов. Формируется набор траекторий путем варьирования конечных условий, а затем из них выбирается траектория, которая лучше всего минимизирует функцию стоимости и, при этом, удовлетворяет ограничениям, например:

$$D_0 = [d_0, \dot{d}_0, \ddot{d}_1] \quad (4)$$

$$D_{1ij} = [d_i, 0, 0, T_j] \quad (5)$$

- где D_0, D_1 — начальное и конечное состояние,
 $d_0, \dot{d}_0, \ddot{d}_0$ — текущие поперечные положение, скорость и ускорение автомобиля,
 d_i — конечное поперечное положение,
 T_j — время выполнения маневра.

Пример планирования поперечных и продольных траекторий представлены на рисунке 9 и 10 соответственно.

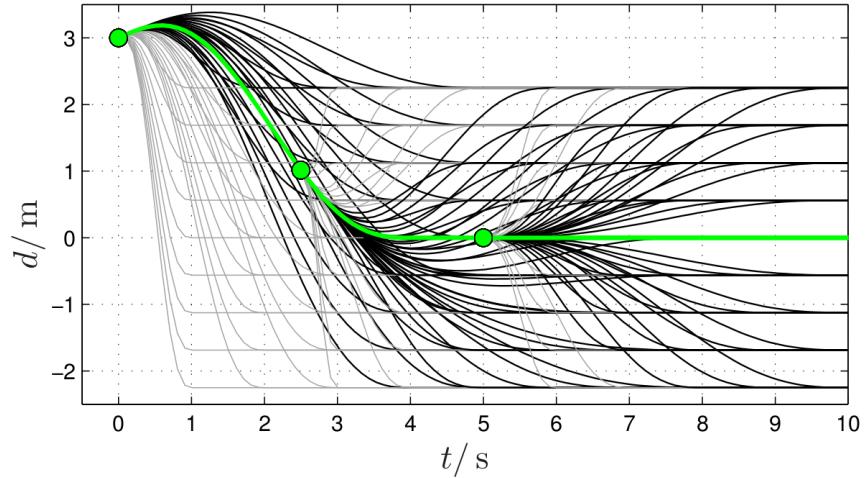


Рисунок 9 – Планирование поперечного движения, зеленая — оптимальная траектория, черная — валидные траектории, серые — невалидные траектории

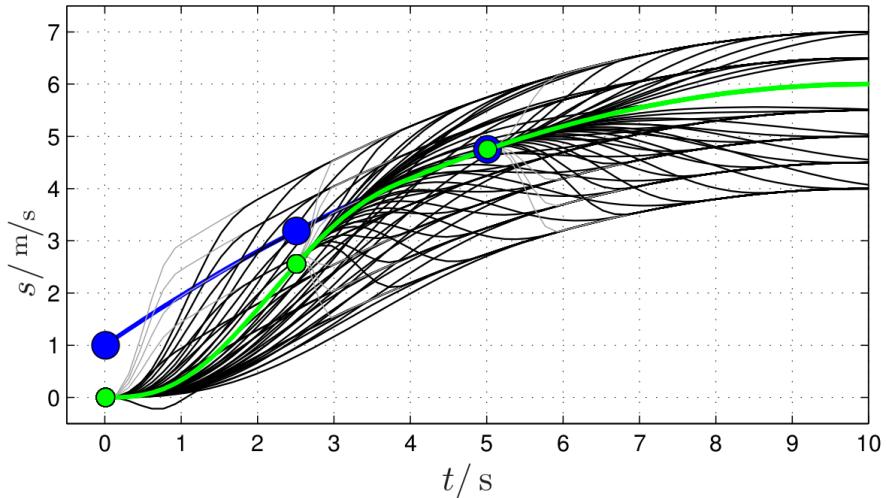


Рисунок 10 – Планирование продольного движения

1.4.2 Команда BOSS в Darpa Urban Challenge

Команда BOSS университета Карнеги-Меллона использовала [**darpa_boss**], [**darpa_boss_2**], [**darpa_boss_3**] несколько иной подход, основанный на методе решеток состояния (state lattice). Планировщик траектории формирует набор траекторий-кандидатов, из которых затем выбирается лучшая, удовлетворяющая ограничениям и функции стоимости.

Каждая траектория-кандидат расчитывается с помощью MPC (Model Predictive Control) генератора [**darpa_boss_kelly**], который формирует динамически достижимый путь между начальным и конечным состояниями. Этот алгоритм может быть использован для формирования набора параметризованных последовательностей управления $[u(p, x)]$, которые удовлетворяют набору ограничений в виде дифференциальных уравнений вида:

$$\dot{x}(x, p) = f(x, u(p, x)) \quad (6)$$

где x - состояние автомобиля (положение x , y , ориентация θ , кривизна траектории k , скорость v), p - набор параметров.

Авторы вводят набор ограничений, которые описываются как разница между целевым состоянием X_c и конечной точкой траектории, получаемой путем интегрирования уравнений динамики автомобиля:

$$x_c = \langle x_c, y_c, \theta_c \rangle^T \quad (7)$$

$$x_f(p, x) = x_i + \int_0^{t_1} \dot{x}(x, p) dt \quad (8)$$

$$C(x, p) = x_c - x_f(p, x) \quad (9)$$

Путем минимизации ошибки $C(x, p)$ находятся такие коэффициенты p , что полученная траектория как можно ближе подходит к заданному конечному состоянию.

Траектории определяются функцией скорости $v(p, t)$ и функцией кривизны $k(p, s)$, где s - покрытая длина дуги. В работе применен ряд различных функций, реализующих различные профили скорости, принимаемые в раз-

ных ситуациях (рисунок 11а). Функция кривизны определяется в виде сплайна второго порядка и задается набором трех опорных точек (рисунок 11б). Каждая траектория определяется набором параметров, которые подбираются в процессе оптимизации.

Траектории расчитываются заранее и формируют lookup-таблицу, с помощью которой определяются подходящие параметры траектории для состояния, близкого к текущему состоянию автомобиля. Это существенно ускоряет операцию планирования движения.

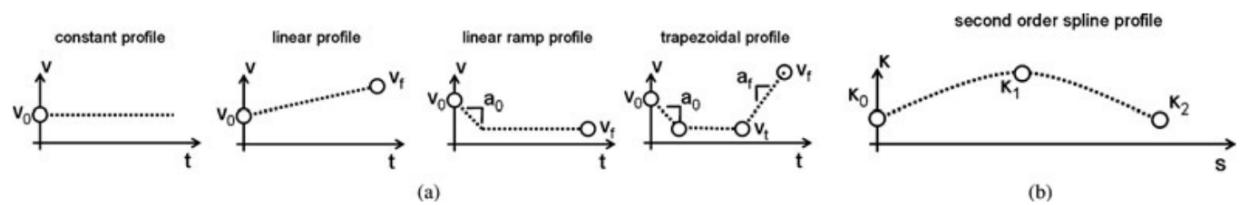


Рисунок 11 – Профили скорости автомобиля BOSS

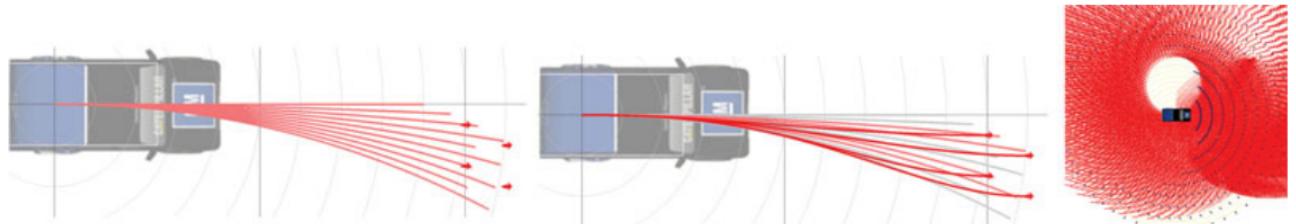


Рисунок 12 – Предварительно расчетанные траектории автомобиля BOSS

1.5 Выводы по главе

Был проведен обзор основных существующих подходов к построению систем управления беспилотными автомобилями, выделены общие принципы архитектуры системы управления, определены типичные компоненты, входящие в ее состав. Были рассмотрены различные методы планирования движения применительно к задаче планирования движения наземного транспортного средства.

2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ УПРАВЛЕНИЯ ДВИЖЕНИЕМ БЕСПИЛОТНОГО АВТОМОБИЛЯ

В данной главе рассматривается этап проектирования структуры и алгоритмов системы управления движением беспилотного автомобиля. Проектирование системы основано на анализе рассмотренных архитектурных решений и алгоритмов.

2.1 Формулирование требований к разрабатываемой системе

Система управления беспилотным автомобилем должна выполнять большое количество функций, необходимых для эффективного и безопасного движения в сложных условиях, среди которых

- распознавание других участников дорожного движения (автомобилей, пешеходов, велосипедистов и т.п.);
- отслеживать других участников дорожного движения в течении некоторого времени для определения характеристик их движения и уменьшения ошибок распознавания;
- определять положение, скорость, ускорение и предсказывать будущие действия других участников дорожного движения;
- распознавать дорожные знаки, светофоры;
- распознавать дорожную разметку;
- на основании всей собранной информации осуществлять планирование безопасного и эффективного движения;
- выполнять запланированное движение и реагировать на изменения в дорожной ситуации.

Исходя из требуемой функциональности, а так же критериев безопасности, система управления беспилотным автомобилем обладает высочайшей сложностью. Разработку такой системы невозможно осуществить в полной мере сразу, по "водопадной" модели. Различные компоненты системы должны разрабатываться и улучшаться независимо, чтобы наращивать функциональность и увеличивать безопасность и эффективность системы управления.

Поэтому главным требованием к системе управления беспилотным автомобилем является высокая модульность, что позволит разрабатывать и тестиировать отдельные системы и подсистемы независимо.

Другим требованием, также приводящим к необходимости модульной системы, является необходимость иметь возможность тестирования компонентов системы управления беспилотным автомобилем на небольших моделях и с использованием различных симуляторов и иных средств моделирования, что позволит быстрее приступить к разработке системы, не завися от реализации системы управления органами управления реального автомобиля.

В связи с очень большим объемом работ, требуемом для разработки полной системы управления беспилотного автомобиля, в рамках данной работы будет осуществляться разработка небольшой части требуемой функциональности:

- осуществление движения по заданной траектории с обратной связью;
- осуществление локального планирования, чтобы формировать оптимальные или близкие к ним траектории движения, позволяющие достичь поставленной локальной цели;
- прототип системы распознавание препятствий, способный распознавать простейшие статические препятствия, для того, чтобы экспериментально проверить предыдущие две возможности;
- управление исполнительными органами мобильной платформы для выполнения запланированных движений.

2.2 Проектирование общей архитектуры системы управления движением

Разрабатываемую в рамках данной работы систему управления можно разделить на несколько частей: драйвер автомобиля, система распознавания препятствий, система планирования локальной траектории и система следования по траектории.

Для отработки и тестирования системы управления необходимо построить небольшую мобильную платформу, которая будет использоваться

для проведения экспериментов. Мобильная платформа должна быть оснащена датчиками, используемыми для распознавания препятствий и определения собственного положения в локальной системе координат, необходимого для реализации движения по траектории с обратной связью. Платформа должна быть оснащена встраиваемым компьютером достаточной производительности. Помимо этого, разумеется, необходимо осуществлять управление приводами платформы.

Для осуществления управления беспилотным автомобилем необходим ряд сенсоров, обеспечивающих систему управления необходимой информацией об окружающей обстановке. Беспилотные автомобили, разрабатываемые крупными компаниями оснащены множество сенсоров, описанных в первой главе. Основными типами сенсоров являются камеры, LiDAR, радары, GPS и другие.

Система компьютерного зрения является одним из самых сложных и критичных компонентов системы управления беспилотного автомобиля, от надежности и работы которой зависит безопасность автомобиля и других участников дорожного движения. Разработка системы компьютерного зрения, которая в полной мере удовлетворяет этим требованиям — крайне сложная и ресурсоемкая задача, над решением которой в течении многих лет работают ведущие компании-разработчики беспилотных автомобилей и, тем не менее, эта задача далека от полного решения.

Все современные, start-of-the-art, системы компьютерного зрения используют глубокие нейронные сети для решения таких задач, как детектирование объектов, детектирование дороги (области доступной для движения), детектирования дорожной разметки, детектирования светофоров и дорожных знаков, предсказания намерений других участников движения. В этих задачах глубокие нейронные сети давно являются стандартом де-факто и существенно превосходят по точности распознавания любые классические алгоритмы. Тем не менее, использование технологий машинного обучения, в частности, глубоких нейронных сетей, приводит к дополнительным трудностям. Помимо разработки архитектуры нейронной сети, поиска ее параметров, крайне важную роль играют данные, используемые для обучения нейронной сети. Именно данные играют критическую роль в том, как система компьютерного зрения будет выполнять свою работу, особенно в сложных

условиях и редких ситуациях. Добиться правильного поведения нейросетевого алгоритма в тех ситуациях, когда он ошибается, можно только с помощью предоставления дополнительных обучающих данных, покрывающих этот случай. Крупные компании, обладающие доступом к огромным массивом данных имеют значительное преимущество в этой области. Так, например, компания Tesla, располагает парком из сотен тысяч автомобилей, которые предоставляют телеметрию, позволяя определять ситуации, в которых работа автопилота была некорректна, собирая похожие ситуации, записанные автомобилями по всему миру, и оперативно дообучать используемые нейронные сети.

В связи с высокой сложностью разработки системы компьютерного зрения, в данной работе не рассматривается разработка полноценной системы компьютерного зрения. Тем не менее, с целью отладки алгоритмов планирования движения и движения по траектории, необходимо разработать простую систему компьютерного зрения, которая бы решала следующие задачи:

- определение положения и ориентации модели автомобиля с точностью, достаточной для работы регулятора с обратной связью для точного движения по траектории;
- определение статических препятствий, с целью проверки системы объезда препятствий.

Для реализации первой задачи было решено применять алгоритм одновременной картографии и навигации (SLAM) на основе стереозрения. SLAM-алгоритмы не позволяют получать глобальное положение автомобиля и подвержены накоплению ошибок при передвижении на большие расстояния, но для проверки задачи локального планирования движения они хорошо подходят, потому что обеспечивают высокую точность позиционирования и частоту обновления данных.

Для определения препятствий было решено использовать шестнадцатилучевой LiDAR Velodyne VLP-16, который был приобретен для проекта беспилотного автомобиля и предполагался для установки на полноразмерный автомобиль. LiDAR позволяет получить трехмерное облако точек, представляющее окружающую обстановку, обладающее сравнительно большой детализацией, обнаруживая препятствия на расстояниях до ста метров. Использование LiDAR для малой мобильной платформы кажется избыточным, но

качественное облако точек, которое он предоставляет, существенно облегчает задачу обнаружения препятствий.

Диаграмма компонентов системы управления беспилотным автомобилем представлена на рисунке 13. Все компоненты можно разделить на следующие группы:

- драйверы сенсоров, осуществляющие работу с сенсорами, такими как LiDAR и камеры, и предоставляющие API для удобного получения данных другими компонентам системы управления;
- основные компоненты системы управления, осуществляющие основную работу по восприятию окружающей обстановки, планированию и управлению движением;
- драйверы исполнительных устройств автомобиля, позволяющие исполнять сформированные последовательности команд непосредственно аппаратным оборудованием автомобиля.

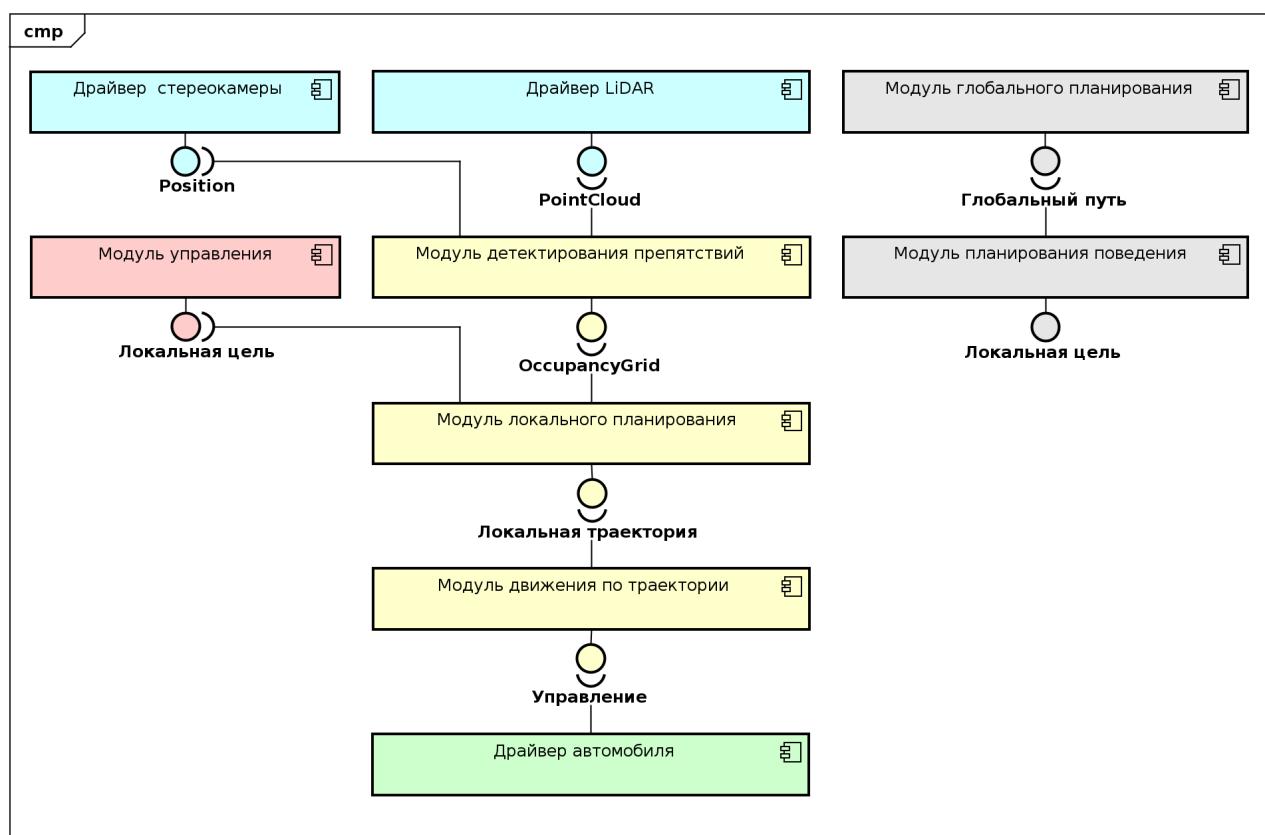


Рисунок 13 – Диаграмма компонентов системы управления беспилотным автомобилем

Драйвер стереокамеры реализует работу со стереокамерой и алгоритм SLAM. Этот модуль предоставляет данные о текущем положении и ориен-

тации автомобиля в локальной системе координат. Драйвер LiDAR представляет облако точек, полученное с LiDAR. Модуль детектирования препятствий осуществляет базовое детектирование препятствий в полученном облаке точек и представляет информацию о них в виде т.н. Occupancy Grid. Occupancy Grid - это распространенный способ представления информации о препятствиях в виде регулярной двухмерной сетки, каждая ячейка которой может быть в одном из трех состояний: свободно, препятствие или неизвестно. Система управления беспилотным автомобилем в общем виде имеет модули глобального планирования и планирования поведения, которые отвечают за построение глобальной траектории по дорожной сети и принятие решений в соответствии с окружающей обстановкой и правилами дорожного движения, как было рассмотрено в первой главе 1. В рамках данной работы эти модули не рассматриваются и заменяются пользовательским интерфейсом оператора, с помощью которого можно вручную указать локальную цель, вместо того, чтобы она задавалась планировщиком поведения. На основании локальной цели, текущего положения автомобиля, полученного от модуля SLAM, и информации о препятствиях, полученных от модуля распознавания препятствий, модуль планирования локальной траектории осуществляет планирование локальной траектории, которая позволит автомобилю достичь (если это возможно), поставленной локальной цели. Сформированный локальным планировщиком путь поступает на вход регулятора с обратной связью, который осуществляет управление исполнительными механизмами автомобиля, чтобы удерживать автомобиль на заданной траектории, получая в качестве обратной связи текущее положение, ориентацию и скорость от модуля SLAM.

2.3 Проектирование подсистемы планирования локальной траектории

Основной системой, разрабатываемой в рамках этой работы, является система планирования локальной траектории. Эта система получает очертную цель от системы планирования поведения и осуществляет формирование оптимальной или близкой к нему достижимой траектории для достижения этой цели. Достижимость подразумевает, что полученная траектория не приведет к столкновению с известными препятствиями, с учетом габаритов препятствий и автомобиля, а также выполнима для автомобиля с точки зрения

его кинематики и динамики движения.

Существует большое количество различных алгоритмов планирования движения, ряд из которых был рассмотрен и проанализирован в первой главе. Наиболее распространенными группами методов являются методы поиска на графах, включая различные модификации, направленные на планирование достижимых для автомобиля путей, такие как методы решетки состояний (state lattice), и методы интерполяции кривых. Первые являются более универсальными, и применяются в том числе для планирования в неструктурированном окружении. Вторые больше ориентированы на планирование движения по дорогам, что накладывает ряд ограничений, упрощающих задачу планирования движения.

В данной рамка данной работы реализуется метод планирования движения по дороге.

2.3.1 Использование алгоритма A*

Первоначальный вариант алгоритма планирования локальных траекторий основывался на алгоритме A*. Причиной выбора этого алгоритма послужило то, что это простой и распространенный алгоритм для нахождения пути на графах, в частности, он широко применяется для работы с Occupancy Grid.

Для работы алгоритм требует опорную траекторию, которая используется в качестве начального приближения, а затем модифицируется, чтобы избежать столкновения с препятствиями.

Таким образом, планировщик поведения или, в случае данной работы, графический интерфейс оператора, должны предоставлять опорную траекторию, помимо требуемого целевого состояния. Это требует дополнительной работы от планировщика поведения и не позволяет сделать интерфейс локального планировщика полностью абстрактным, не зависящим от его реализации, потому что не всем возможным реализациям требуется опорная траектория (например, алгоритмам на основе RRT она не требуется). Тем не менее, это не является значительным препятствием. Опорная траектория при движении по непрерывному сегменту дороги может быть реализована как центральная линия текущей полосы, по которой движется автомобиль. Это

может быть определено с помощью системы компьютерного зрения. Для более сложных ситуаций, например, перекрестков, подобная траектория может быть реализована в виде сплайна, соединяющего соответствующие полосы двух пересекающихся дорог.

Алгоритм работает следующим образом. Рассматривается ближайший участок опорной траектории определенной длины, и если на этому участке опорная траекторий пересекается с препятствием, то запускается алгоритм обхода препятствия, в противном случае продолжается движение по опорной траектории. Для обхода препятствия ищется точка на опорной траектории, где заканчивается препятствие (в случае, если это нельзя определить, например, препятствие закрывает обзор системе компьютерного зрения, берется некий допуск от последних видимых клеток препятствий на Occupancy Grid). После этого запускается алгоритм A* для нахождения пути от текущего положения автомобиля до точки выхода опорной траектории из зоны препятствия. Алгоритм запускается с определенной периодичностью и перестраивает траекторию, таким образом, по мере движения автомобиля системе компьютерного зрения становятся обозримы новые участки препятствия и траектория перестраивается.

Этот алгоритм обладает серьезными недостатками:

- не учитывает кинематику и динамику автомобиля;
- не планирует скорость движения.

Алгоритм был реализован и испытан на мобильной платформе. По результатам испытаний были выявлены дополнительные недостатки алгоритма. В связи с неидеальностью системы компьютерного зрения, при обновлении периодическом Occupancy Grid происходит небольшое "мерцание" т.е. те или иные пограничные ячейки карты могут менять свое состояние. Это приводит к тому, что алгоритм резко перестраивает траекторию движения. Особенно это было заметно, когда модель приближалась к симметричному препятствию. В таком случае, алгоритм мог прокладывать обходную траекторию то по левую сторону от препятствия, то по правую сторону от препятствия. Это приводило к тому, что модель не могла корректно осуществить маневр и врезалась в препятствие.

По результатам испытаний было выявлено, что рассмотренный алгоритм не подходит для локального планирования траекторий движения бес-

пилотного автомобиля.

2.3.2 Метод интерполяции кривых

Исходя из результатов испытания алгоритма на основе A*, было принято решение разработать другой алгоритм. За основу для проектирования системы планирования локальной траектории был выбран метод, предложенный командой "Junior"Darpa Urban Challenge [**darpa_junior**], который был кратко рассмотрен в главе 1. Преимуществами этого подхода являются:

- лучший учет кинематических и динамических ограничений автомобиля за счет планирования траектории в виде гладких кривых;
- применение полиномов пятого порядка позволит планировать гладкие траектории с гладкой первой производной (скоростью), благодаря чему не будет осуществляться резкая перестройка траектории;
- алгоритм позволяет находить оптимальные или близкие к оптимальным траектории, в отличие от, например, алгоритмов на основе RRT;
- алгоритм обладает меньшей вычислительной сложностью, чем алгоритмы на основе RRT.

Основным недостатком данного алгоритма является то, что в нем нет настоящего учета динамической модели автомобиля. Для учета динамических ограничений применяются косвенные признаки, такие как максимальные продольные скорость и ускорение, максимальное боковое ускорение, минимальный радиус кривизны, которые могут быть получены в результате анализа динамической модели автомобиля. Для обеспечения соблюдения этих условий в широком диапазоне различных условий приходится устанавливать эти ограничения с запасом. Это приводит к тому, что динамические возможности автомобиля используются не полностью. Нарушение динамических ограничений также не исключено, например, при смене условий (например, мокрая дорога), что может снизить безопасность автомобиля. Тем не менее, при выборе достаточно жестких ограничений, можно обеспечить высокую надежность этого алгоритма. Алгоритм хорошо зарекомендовал в соревновании DARPA Urban Challenge [**darpa_junior**].

Для работы методы необходимы следующие входные данные:

- опорная (референсная) траектория;

- локальная цель, описывающая требуемое положение и скорость автомобиля, эта точка должна лежать на опорной траектории;
- текущее состояние автомобиля (положение, скорость, ускорение);
- карта препятствий в формате Occupancy Grid.

Этот алгоритм, как и предыдущий, требует получения от планировщика поведения опорной траектории, помимо целевого состояния.

Планирование траектории осуществляется независимо для продольного движения вдоль опорной траектории и поперечного движения. Планирование осуществляется в подвижной системе координат, движущейся по опорной траектории вместе с автомобилем, как показано на рисунке 14.

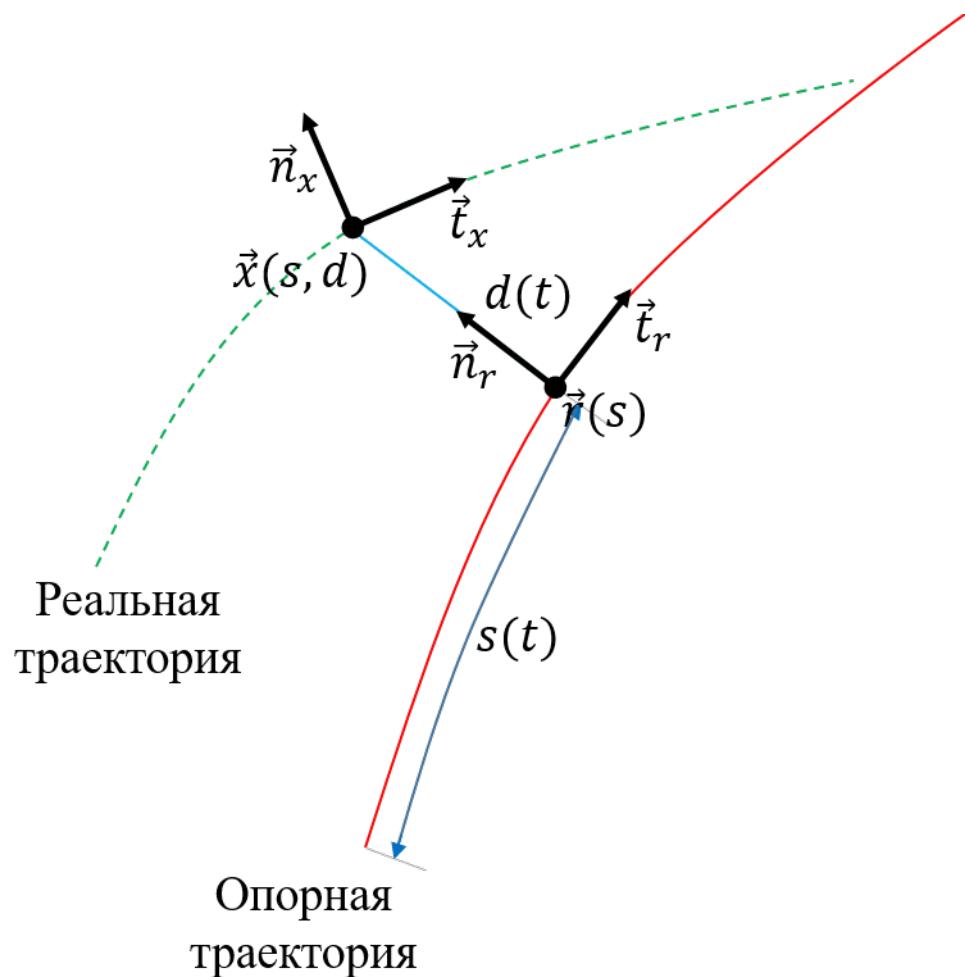


Рисунок 14 – Диаграмма компонентов системы управления беспилотным автомобилем

Опорная траектория (изображена на рисунке красной линией) представлена натурально параметризованная кривая, где $s(t)$ — покрытая длина дуги в момент времени t . Положение автомобиля в глобальной системе координат обозначено радиус-вектором \vec{x} . Этому положению автомобиля соответ-

стует подвижная система координат $\vec{r}(s)$, представляющая собой ортонормированный репер Френе, где \vec{t}_r и \vec{n}_r — касательный и нормальный вектора к опорной траектории соответственно. Таким образом, положение автомобиля в глобальной декартовой системе координат (x, y) может быть представлено в подвижной системе координат как (s, d) , где d — расстояние между автомобилем и опорной траекторией.

Продольная и поперечная траектории движения ищутся в форме полиномов пятого порядка, соединяющие начальное состояние (текущее состояние автомобиля, полученное от сенсоров) и целевое состояние (полученное от планировщика поведения или графического интерфейса оператора). Первые и вторые производные этих полиномов представляют собой зависимость скорости и ускорения автомобиля от времени, которые затем подаются в регулятор с обратной связью наряду с пространственной траекторией движения, т.е. этот метод позволяет планировать не только траекторию перемещения автомобиля, но и профиль скорости.

Задача состоит в том чтобы найти коэффициенты полинома, которые минимизируют некую функцию стоимости, используемую для оценки траекторий. За основу для функции стоимости взят интеграл производной ускорения по времени \ddot{s} , называемая рывок (jerk). Это является распространенным выбором функции стоимости во многих алгоритмов планирования движения автомобилей. Ниже представлены интегралы для продольного и поперечного движения соответственно:

$$J_s = \int_{t_0}^{t_1} \ddot{s}(t)^2 dt \quad (10)$$

$$J_d = \int_{t_0}^{t_1} \ddot{d}(t)^2 dt \quad (11)$$

где $s(t), d(t)$ — функция продольной и поперечной траекторий соответственно,

t_0, t_1 — начальное и конечное время маневра соответственно.

Такой выбор функционала стоимости обоснован следующим:

- минимизация количества и резкости маневров, совершаемых автомобилем (резкая траектория с большим количеством маневров может

быть получена в результате других алгоритмов планирования, таких как RRT), что повышает безопасность и экономичность движения;

- обеспечение комфорта пассажиров

В соответствии [darpa_junior_frenet_origin] оптимальное решение, минимизирующее рывок, может быть найдено в форме полиномов пятого порядка:

$$s(t) = a_0t^5 + a_1t^4 + a_2t^3 + a_3t^2 + a_4t + a_5 \quad (12)$$

$$\dot{s}(t) = 5a_0t^4 + 4a_1t^3 + 3a_2t^2 + 2a_3t + a_4 \quad (13)$$

$$\ddot{s}(t) = 20a_0t^3 + 12a_1t^2 + 6a_2t + 2a_3 \quad (14)$$

Помимо рывка, функционал стоимости должен содержать ряд других членов:

- отклонение конечной точки поперечной траектории от опорной траектории — эта функция стоимости ухудшает оценку поперечных траекторий, которые не достигают заданной цели;
- отклонение конечной точки продольной траектории от покрытой длины дуги целевой точки — эта функция стоимости ухудшает оценку продольных траекторий, которые не достигают заданной цели;
- отклонение первой производной продольной траектории от требуемой скорости — эта функция стоимости ухудшает оценку продольных траекторий, которые не достигают заданной скорости;
- общее время совершения маневра — эта функция стоимости ухудшает оценку слишком долгих траекторий.

Полный функционал стоимости для поперечных траекторий:

$$C_d = K_{dj} \int_0^T \ddot{d}(t)^2 dt + K_d d(T)^2 + K_{dt} T \quad (15)$$

Полный функционал стоимости для продольных траекторий:

$$C_s = K_{sj} \int_0^T \ddot{s}(t)^2 dt + K_s(s(T) - S_1)^2 + K_v(\dot{s}(T) - \dot{S}_1)^2 + K_{st} T \quad (16)$$

где $K_{dj}, K_d, K_{dt}, K_{sj}, K_s, K_v, K_{st}$ — весовые коэффициенты,
 S_1 — конечное (целевое) продольное со-
стояние,
 T — время выполнения маневра.

Итоговый функционал стоимости для траектории представляет собой взвешенную сумму стоимостей продольного и поперечного движения:

$$C_{tot} = K_{lon} + C_s + K_{lat} + C_d \quad (17)$$

Проблема заключается в том, что при оптимизации коэффициентов полинома необходимо учитывать ограничения. На траекторию накладываются следующие ограничения:

- максимальная продольная скорость,
- максимальное продольное ускорение (и замедление),
- максимальное поперечное ускорение,
- минимальная кривизна траектории,
- отсутствие пересечения с препятствиями.

Осуществление оптимизации коэффициентов полинома с учетом необходимости проверки на отсутствие пересечений с препятствиями, которые представлены в виде Occupancy Grid. Поэтому вместо применения методов оптимизации применяется широко распространенный в задачах планирования движения прием — генерируется большой набор траекторий путем варьирования конечного состояния $S_1(T) = < s(t), \dot{s}(T), \ddot{s}(T) >$ и времени T в некотором диапазоне, а затем производится выбор траектории с наименьшим значением функции стоимости среди тех, которые удовлетворяют ограничениями.

Имея начальное состояние $< s(0), \dot{s}(0), \ddot{s}(0) >$, конечное состояние $< \dot{s}(T), \ddot{s}(T), \ddot{s}(T) >$ и время совершения маневра T , можно рассчитать ко-

эффициенты полинома, решив систему уравнений:

$$\left\{ \begin{array}{l} a_5 = s(0) \\ a_4 = \dot{s}(0) \\ 2a_3 = \ddot{s}(0) \\ a_0 T^5 + a_1 T^4 + a_2 T^3 + a_3 T^2 + a_4 T + a_5 = s(T) \\ 5a_0 T^4 + 4a_1 T^3 + 3a_2 T^2 + 2a_3 T + a_4 = \dot{s}(T) \\ 20a_0 T^3 + 12a_1 T^2 + 6a_2 T + 2a_3 = \ddot{s}(T) \end{array} \right. \quad (18)$$

На рисунке 15 представлены графики примера продольной траектории, полученные путем решения системы 18. В этом примере автомобиль тормозит с начальной скорости 15 м/с до полной остановки за 7 секунд, преодолевая 60 метров.

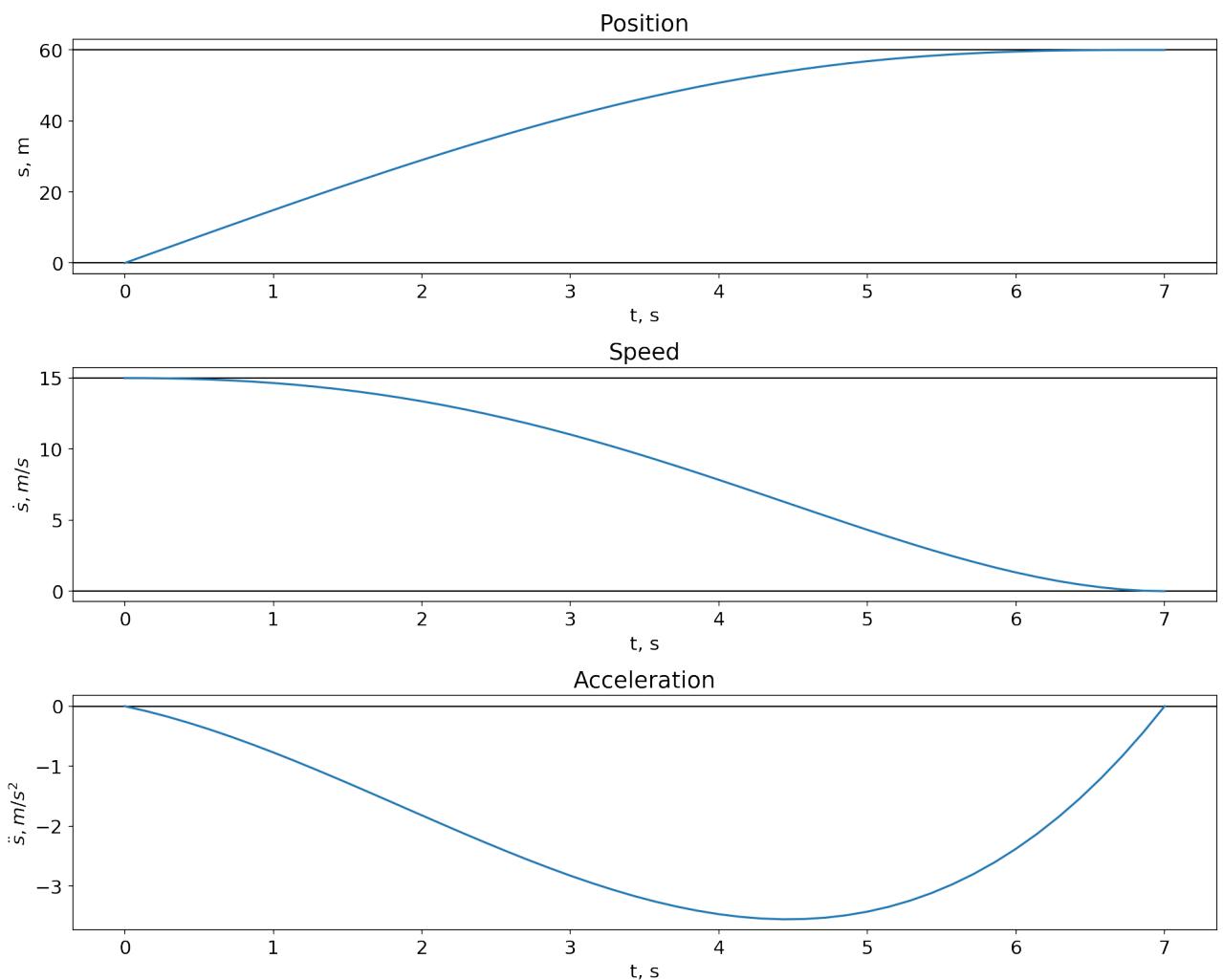


Рисунок 15 – Пример траектории, описываемой полиномом пятого порядка

Набор граничных условий, необходимый для формирования набора траекторий-кандидатов, формируется следующим образом. Начальное продольное состояние:

$$S_0 = \langle s_c, \dot{s}_c, 0 \rangle \quad (19)$$

где s_c — текущее положение автомобиля вдоль опорной траектории, полученное от системы SLAM,
 \dot{s}_c — текущая продольная скорость автомобиля, полученная от системы SLAM.

Начальное поперечное состояние:

$$D_0 = \langle d_c, \dot{d}_c, 0 \rangle \quad (20)$$

где d_c — текущее поперечное отклонение автомобиля от опорной траектории, полученное от системы SLAM,
 \dot{d}_c — скорость изменения поперечного отклонения, полученная от системы SLAM.

Стоит заметить, что поперечная скорость \dot{d}_c в общем случае не является скоростью бокового скольжения автомобиля и может быть отличной от нуля даже в том случае, если автомобиль движется без бокового скольжения. Если вектор продольной скорости автомобиля не параллелен касательной к опорной траектории, вектор продольной скорости будет разложен в пару векторов \dot{s}_c и \dot{d}_c .

Конечное продольное состояние:

$$\begin{aligned} S_{1_{ij}} &= \langle s_i, v_j, 0 \rangle \\ s_i &= s_1 + i\Delta s \\ v_j &= \dot{s}_1 + i\Delta v \end{aligned} \quad (21)$$

где s_1 — требуемое положение автомобиля вдоль опорной траектории, получаемое от планировщиков высокого уровня,
 \dot{s}_1 — требуемая продольная скорость автомобиля, получаемая от планировщиков высокого уровня,
 Δs — шаг перебора продольных положений,
 Δv — шаг перебора продольных скоростей.

Конечное поперечное состояние:

$$\begin{aligned} D_{1_{ij}} &= \langle d_i, 0, 0 \rangle \\ d_i &= i\Delta d \end{aligned} \tag{22}$$

где Δd — шаг перебора поперечных положений.

Для перебора конечных состояний поперечных траекторий принимается, что требуемое поперечное положение (отклонение автомобиля от опорной траектории) всегда равно нулю, т.е. автомобиль должен двигаться ровно по опорной траектории. В том случае, когда это невозможно, в силу наличия препятствий, или существует более оптимальная траектория (например, обгон медленно движущегося спереди автомобиля), поперечное движение автомобиля в пределах полосы или выход за ее пределы осуществляется путем варьирования конечных поперечных состояний в локальном планировщике движения, а не за счет передачи соответствующих целевых положений планировщиками движения более высокого уровня. Планировщик движения более высокого уровня, такой как планировщик поведения, может дать команду на смену полосы движения, путем указания новой опорной траектории, а требуемое поперечное отклонение от новой траектории по прежнему будет нулевым.

Значения требуемых поперечных скоростей и ускорения полагаются всегда равными нулю, т.е. ожидается, что после завершения маневра автомобиль будет двигаться параллельно опорной траектории.

Для расчета коэффициентов полинома, описывающего траекторию, требуется определение начального и конечного состояний, как было описано выше. В качестве начального состояния принимается текущее состояние автомобиля, получаемое от системы одновременной картографии и навигации (SLAM). Как было сказано выше, система SLAM, позволяет получить текущее положение $\langle x, y, z \rangle$ и ориентацию в виде кватерниона углов Эйлера $\langle \psi, \theta, \varphi \rangle$, где ψ — рыскание (yaw), вращение вокруг оси z ,

θ — тангаж (pitch), вращение вокруг y ,

φ — крен (roll), вращение вокруг оси x .

Углы Эйлера и соответствующие оси изображены на рисунке 16.

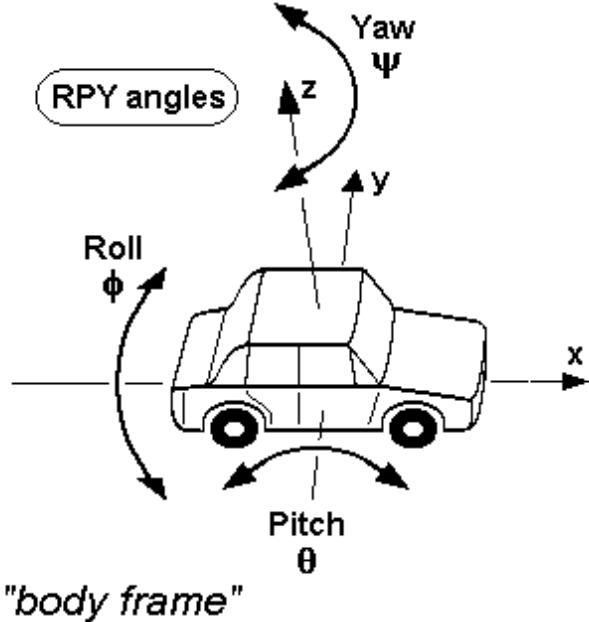


Рисунок 16 – Описание вращения с помощью углов Эйлера

Таким образом, для автомобиля, движущегося на плоскости, состояние может быть задано следующим образом: $\langle x, y, \psi, \dot{x}, \dot{y}, \dot{\psi}, \ddot{x}, \ddot{y}, \ddot{\psi} \rangle$. Принимая, что боковое скольжение отсутствует, состояние автомобиля можно записать в виде $\langle x, y, \psi, v, \dot{\psi}, a, \ddot{\psi} \rangle$, где v — продольная скорость, a — продольное ускорение.

Для применения рассматриваемого алгоритма планирования движения, требуется перевести состояние автомобиля из глобальной декартовой системы координат в подвижную систему координат Френе.

Помимо начального $\langle s_0, \dot{s}_0, \ddot{s}_0 \rangle$, и конечного $\langle s_1, \dot{s}_1, \ddot{s}_1 \rangle$ состояний, для расчета коэффициентов полинома, согласно 18 требуется задать требуемое время маневра T . С точки зрения определения движения автомобиля, явное задание скорости, пройденного расстояния и требуемого для этого времени, является избыточным. В большинстве случаев требуется монотонное изменение скорости автомобиля (например, ускорение или замедление). Например, рассмотрим пример, приведенный на рисунке 15, где автомобиль останавливается с начальной скорости 15 м/с до полной остановки за 7 секунд, преодолевая 60 метров. Для определения этой траектории достаточно двух параметров: начальная скорость и требуемый тормозной путь позволяют рассчитать требуемое время, а начальная скорость и требуемое время до полной остановки позволяют рассчитать требуемый тормозной путь. Существенное превышение или уменьшение времени маневра T по сравнению с необходимым

для совершения этого маневра, не позволит получить монотонную гладкую траекторию.

На рисунке 17 представлены три траектории: голубая сплошная с близкой к оптимальной длительности маневра, оранжевая пунктирная со слишком малой длительностью маневра и зеленая штрихпунктирная со слишком большой длительностью маневра. Видно, что при оптимальной длительности маневра положение автомобиля монотонно возрастает, а скорость — монотонно убывает. При слишком малой длительности маневра первоначально происходит ускорение и лишь потом замедление до требуемой скорости. При слишком большой длительности маневра автомобиль сначала проезжает мимо целевого положения, а затем возвращается, т.е. имеете место участок с отрицательной скоростью (движение назад).

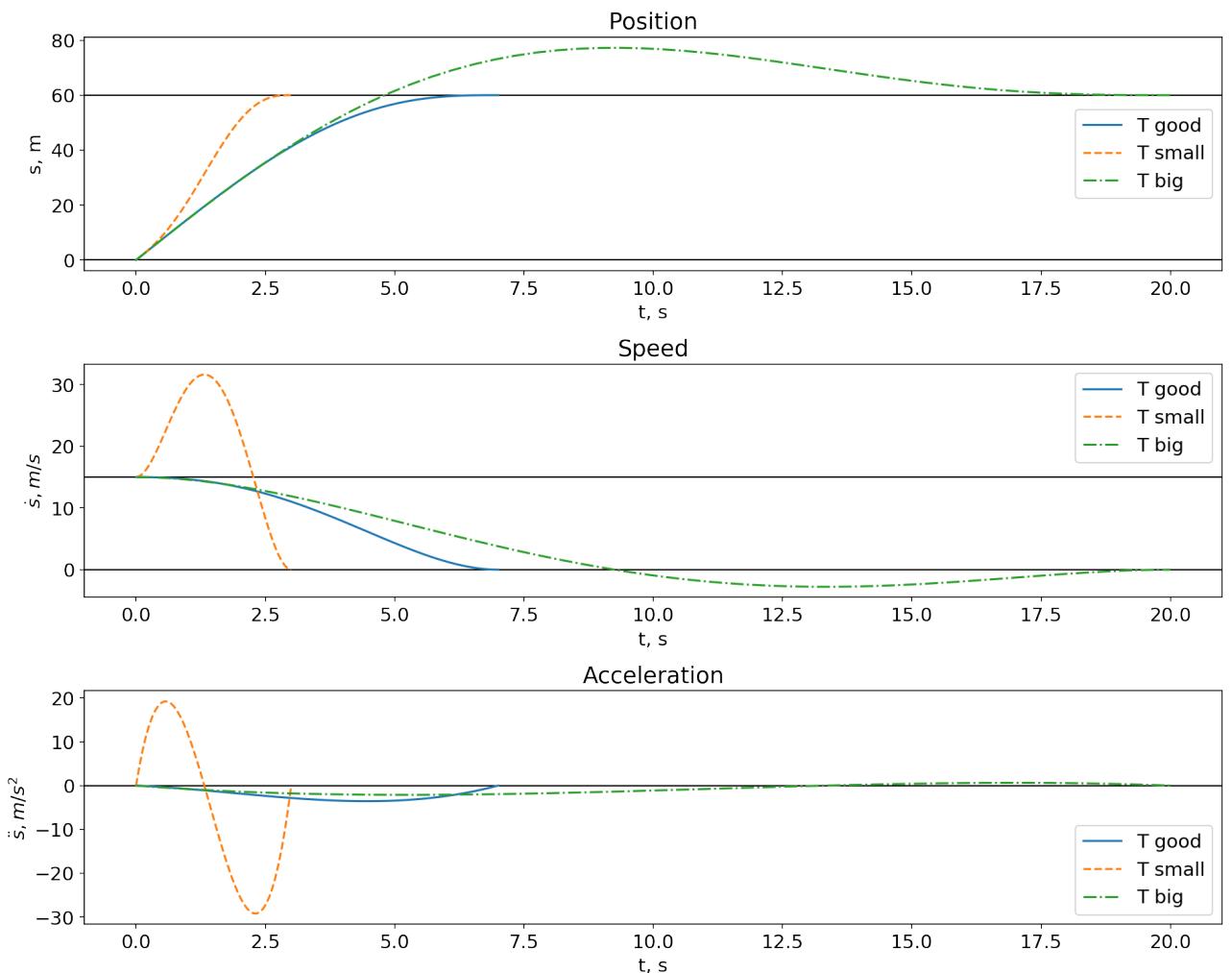


Рисунок 17 – Зависимость траектории от времени совершения маневра

Варьирование времени маневра T может быть применено для получения различных профилей скорости. Так, например, первоначальное ускоре-

ние с последующим замедлением может быть использовано при маневрах обгона. В работе [**darpa_boss**], посвященной автомобилю "BOSS"Darpa Urban Challenge применяются несколько явно заданных профилей скорости. С одной стороны, явное параметрическое задание профиля скорости позволяет более точно и задавать требуемый профиль скорости и иметь большее число вариантов профиля скорости, в отличие от примененного в этой работе метода, в котором профиль скорости задается неявно путем варьирования одного параметра — длительности маневра T . С другой стороны, метод, применяемый в автомобиле BOSS, обладает следующим недостатком: форма траектории и профиль скорости выбираются независимо, в то время как в методе, реализуемым в данной работе, профили скорости и ускорения являются производными траектории, таким образом, этот метод лучше учитывает динамические характеристики автомобиля.

Таким образом, аналогично с варьированием конечных состояний, требуется варьировать длительность маневра:

$$T_k = T_e + k\Delta T \quad (23)$$

где T_e — начальная оценка длительности маневра,

ΔT — шаг перебора длительностей маневра.

Можно было бы задать в качестве начальной оценки заранее определенное константное значение времени либо перебирать все значения от 0 до некого заданного T_{max} . Но это было бы неоптимально, потому что в таком случае было бы сгенерировано большое количество траекторий со слишком малой или слишком большой длительностью маневра. Эти траектории не влияют на итоговый результат планирования движения, т.к. в любом случае будет выбираться оптимальная траектория, минимизирующая функционал стоимости, но расчет большого количества избыточных траекторий серьезно увеличит вычислительные затраты и время работы алгоритма.

Чтобы избежать этого был предложен метод приблизительной оценки требуемой длительности маневра T_e . В качестве оценки требуемого времени совершения маневра принимается время совершения маневра с такими же начальными и конечными условиями, но для случая равноускоренного дви-

жения:

$$\begin{cases} \dot{s}_1 = \dot{s}_0 + aT_e \\ s_1 = s_0 + \dot{s}_0 T_e + \frac{aT_e^2}{2} \end{cases} \quad (24)$$

Решая эту систему получим формулу расчет оценки времени маневра:

$$T_e = 2 \frac{s_0 - s_1}{\dot{s}_0 + \dot{s}_1} \quad (25)$$

На рисунке 18 приведен пример оценки длительности маневра. Пунктирная линия — траектория равноускоренного движения, сплошные линии — траектории, полученные путем варьирования длительности маневра при одинаковых начальных и конечных условиях. Данная оценка не является точной, потому что траектория движения ищется в форме полиномов пятого порядка, соответственно, профиль скорости описывается полиномами четвертого порядка, а предложенный метод оценивает необходимое время исходя о предположении о том, что ускорение неизменно и скорость представлена в виде линейной функции. Тем не менее, эксперименты показали, что эта оценка подходит для практического использования, т.к. после нахождения начального приближения длительности маневра формируется ряд траекторий-кандидатов путем варьирования в том числе длительности маневра, из которых затем будет выбрана траектория, минимизирующая функционал стоимости, поэтому даже если начальное приближение было не оптимальным, может быть выбрана более оптимальная траектория. Варьирование длительности маневра создает при прочих неизменных параметрах создает набор траекторий-кандидатов, отличающихся профилем скорости. Различные профили скорости играют роль при взаимодействии с препятствиями, когда в результате совокупного влияния различных функционалов стоимости, оптимальный выбор может отличаться от случая движения без помех, для которого оптимальным вариантом будет выбор траектории, наиболее приближенной к начальной оценке. Для такого случая можно было бы дать точную оценку длительности маневра, которое минимизирует функционал стоимости, но в более общем случае преимущества такой оценки сходят на нет. Преимуществом предложенной оценки является ее крайне низкая вычислительная сложность, что немаловажно, т.к. в процессе планирования движения требуется рассчитывать большое количество траекторий и осуществлять перепла-

нирование как можно чаще.

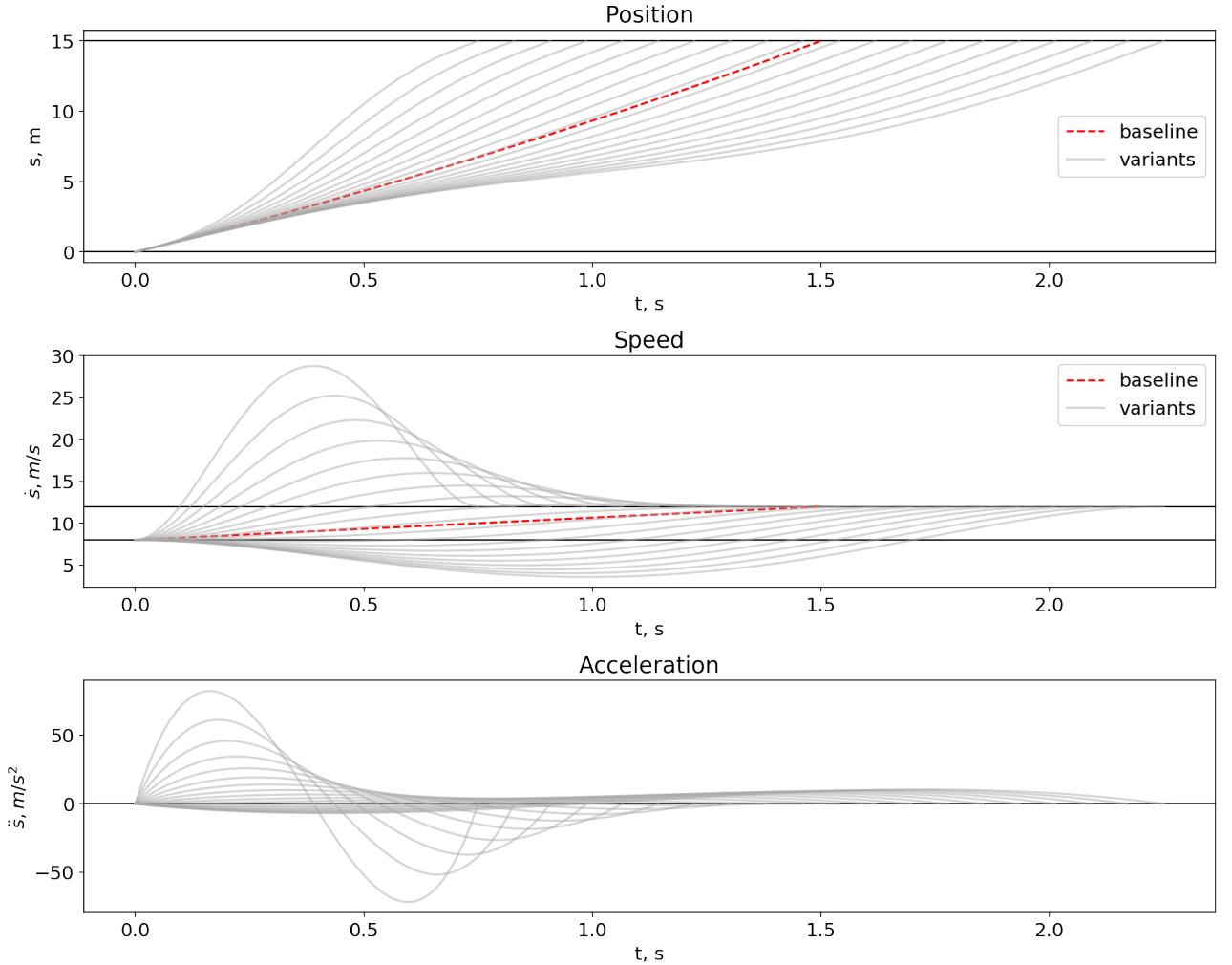


Рисунок 18 – Пример оценки длительности маневра с помощью равноускоренного движения. Пунктирная линия — траектория равноускоренного движения, сплошные линии - траектории, полученные путем варьирования длительности маневра при одинаковых начальных и конечных условиях.

Блок-схема рассмотренного алгоритма приведена на рисунке 19.

На вход алгоритма подаются состояние автомобиля, полученное от системы SLAM, карта препятствий и локальная цель. Вначале происходит преобразование начального состояния и целевого состояния в подвижную систему координат Френе, заданную опорной траекторией. На основе заданных s_0 , \dot{s}_0 , s_1 , \dot{s}_1 , происходит расчет оценочного времени маневра T_e .

Далее осуществляется перебор значение t_i , s_i и v_i во вложенных циклах. Для того, чтобы было возможно совместить продольные и поперечные траектории необходимо, чтобы эти коэффициенты этих траекторий были рассчитаны для одинаковой длительности маневра t_i и значения этих функций

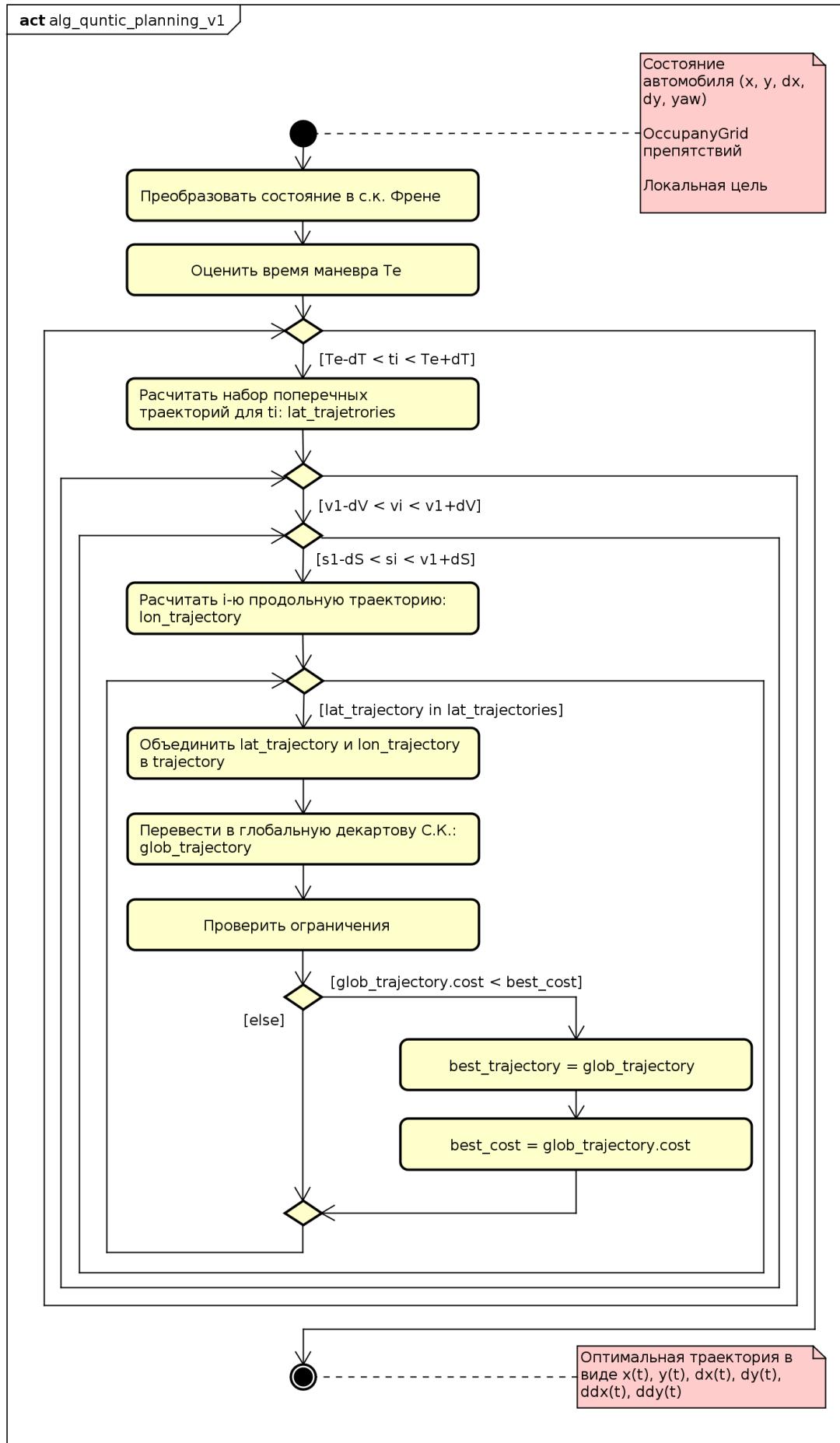


Рисунок 19 – Блок-схема алгоритма планирования движения

табулированы одинаковыми набором значений t .

Так как расчет продольных и поперечных траекторий независим, алгоритм можно оптимизировать, рассчитывая набор траекторий один раз для каждого t_i , а затем объединяя их. Таким образом, на соответствующем шаге алгоритма для каждого t_i генерируется и сохраняется набор поперечных траекторий, а затем для каждого s_i и v_i генерируется очередная продольная траектория, которая объединяется со всеми предварительно рассчитанными поперечными траекториями. Полученная траектория переводится обратно в глобальную систему координат и проверяется на удовлетворение ограничениям, таким как минимальная кривизна и отсутствие пересечений с препятствиями.

На рисунке 20 представлен пример набора траекторий, сформированных алгоритмом путем варьирования целевого продольного положения, целевого поперечного положения и длительности маневра. Параметры работы алгоритма выбраны для большей наглядности полученного изображения, и отличаются от реальных параметров, которые необходимо использовать при работе алгоритма. На рисунке представлены набор поперечных (слева) и продольных (справа) траекторий, включая их первые и вторые производные, а также объединенные траектории. Цветом обозначается результат проверки ограничений. Серым обозначены траектории, которые не удовлетворяют ограничениям, красным - траектории, которые удовлетворяют ограничениям.

2.3.3 Планирование на несколько шагов

Приведенный выше алгоритм был реализован и испытан на мобильной платформе с использованием обнаружения препятствий с помощью LiDAR. Результаты испытания алгоритма показали, что алгоритм обладает недостатком, приводящим в определенных ситуациях к тому, что он не может найти допустимую траекторию и автомобиль не может продолжать движение. Иллюстрация такой ситуации приведена на рисунке 21.

На рисунке изображена смоделированная ситуация. Автомобиль движется по дороге с постоянной скоростью, имеется локальная цель с продольной координатой 40 м. Препятствие в форме окружности радиусом 1.5 м имеет продольную координату 30 м. Для наглядности изображения осуществ-

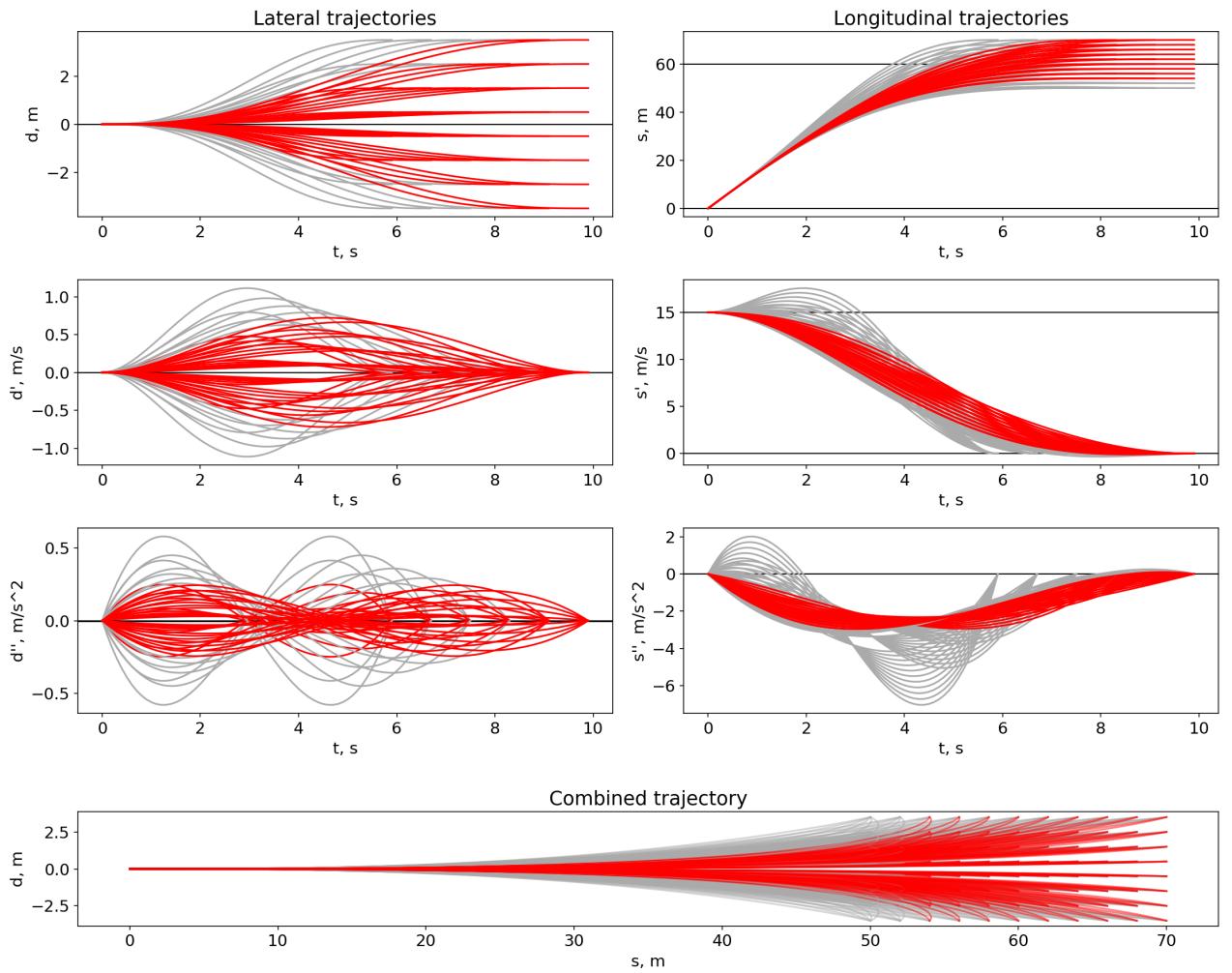


Рисунок 20 – Пример формирования набора траекторий кандидатов. Серым обозначены траектории, которые не удовлетворяют ограничениям, красным – траектории, которые удовлетворяют ограничениям

лялось варьирование только продольных и поперечных координат конечного состояния, скорость и время маневра не изменялись. На рисунке изображены два результата планирования с различными весовыми коэффициентами в функционале стоимости.

Для того, чтобы объехать препятствие, автомобиль должен отклониться от опорной траектории. Этот маневр приведет к следующим изменениям в функционале стоимости 16:

- отклонение d от опорной траектории увеличится, что приведет к увеличению члена $K_d d(T)^2$;
- совершенные маневры приведут к увеличению $K_{dj} \int_{t_0}^{t_1} \ddot{s}(t)^2 dt$;
- общее время маневра приведет к увеличению члена $K_{st} T$;
- автомобиль будет ближе к продольной цели, что приведет к умень-

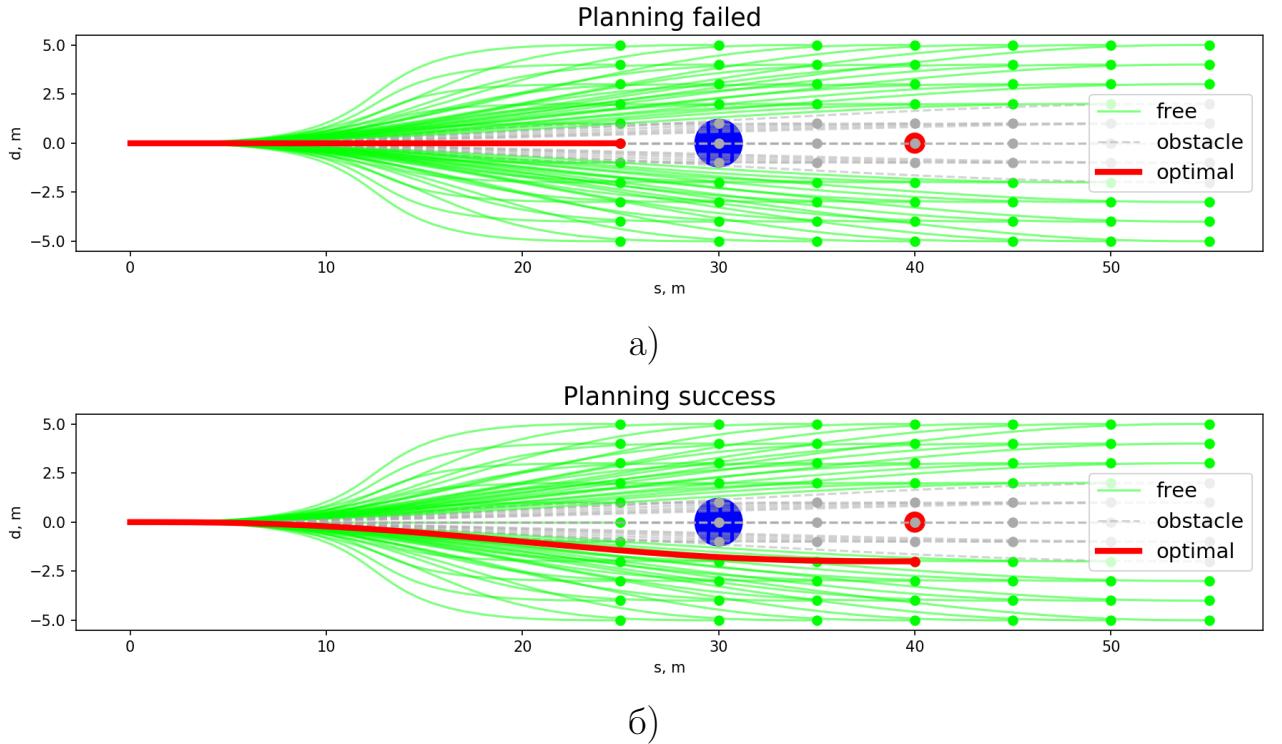


Рисунок 21 – Пример планирования траектории с учетом препятствия при различных весовых коэффициентах: а) подходящая траектория не была найдена, б) подходящая траектория была найдена

шению $K_s(s(T) - S_1)^2$

Итоговое значение функционала стоимости и принятное решение зависит от соотношения весовых коэффициентов K_d , K_{dj} , K_{st} , K_s .

Эту проблему можно решить, если модифицировать алгоритм планирования движения путем осуществления нескольких шагов планирования, т.е. рекурсивно запустить планирование повторно из каждого конечного состояния. Такой метод существенно увеличит вычислительную сложность алгоритма, но позволит осуществлять планирование с большим горизонтом и осуществлять более сложные маневры.

Для реализации этого алгоритма было решено совместить рассмотренный выше алгоритм с механизмом решетки состояний (state lattice) [**motion_planning_lattice_1**]. Как было рассмотрено в первой главе, эквивалентные траектории можно представить в виде последовательности элементарных траекторий. В данном случае, требуется заменить рекурсивный вызов планирования из каждой конечной точки построением графа, куда каждая начальная и конечная точка входит только один раз. Дело в том, что из-за того, что конечные состояния представлены в виде регулярной ре-

шетки, попасть в некое конечное состояние можно несколькими способами, и осуществлять планирование отрезка траектории, начинающегося из этого состояния несколько раз не требуется. Это позволит существенно уменьшить вычислительную сложность алгоритма.

Помимо реализации планирования на несколько шагов вперед, в алгоритм добавлена еще одна модификация. В предыдущей версии алгоритма оценка длительности времени маневра осуществляется на основе целевого положения, а затем используется при всех перебираемых значениях конечного положения и скорости. В новой версии алгоритма оценка времени осуществляется для каждой пары начальное состояние-конечное состояние. Сравнение двух методов формирования траекторий приведено на рисунке 22. При оценке времени маневра для каждой пары состояний, все полученные траектории удовлетворяют требованию монотонности, т.е. в рассматриваемом примере торможения автомобиля не происходит ни ускорений, ни возврата назад, как это происходит в первом варианте. Для получения различных профилей скорости, применяется варьирование длительности маневра уже после оценки.

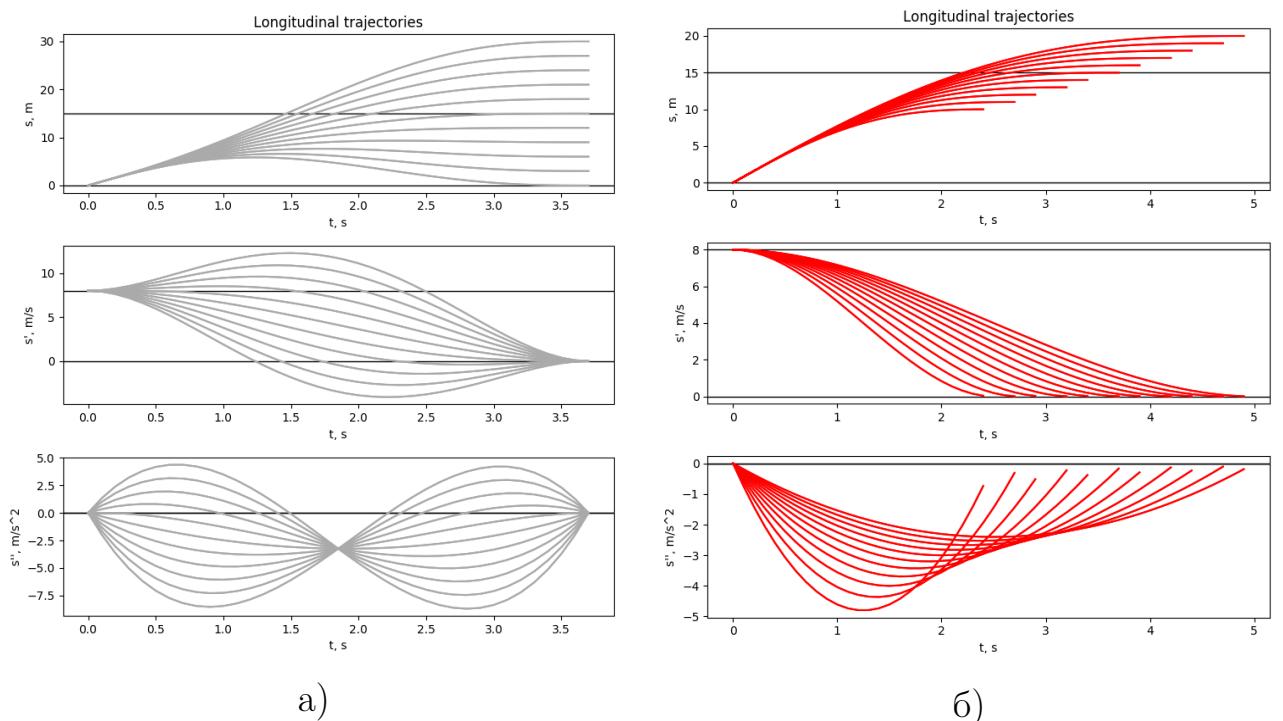


Рисунок 22 – Сравнение получаемых траекторий при оценке времени: а) при оценке времени один раз на основе цели, б) при оценке времени для каждой пары "начальное состояние-конечное состояние"

Планирование траекторий осуществляется путем построения графа возможных траекторий и нахождении оптимальной траектории с помощью

алгоритма Дейкстры.

Алгоритм Дейкстры является хорошо известным алгоритмом, для решения т.н. задачи "Single-source Shortest Path т.е. нахождение кратчайших путей на графе до всех вершин от заданной. В отличие от этой задачи, в задаче планирования движения требуется не только найти кратчайший путь на графике, но и выбрать конечную вершину, которая даст оптимальную траекторию движения. Исходя из функционала стоимости 17, применяемого для оценки траекторий кандидатов, можно сказать, что оценка траектории складывается из двух компонент: стоимости траектории, определяемой резкостью совершаемых маневров, и стоимостью конечного состояния, определяемого его удаленности от цели. С учетом этого, можно переписать функционал стоимости следующим образом, просто перегруппировав члены и разделив его на функционал стоимости движения и функционал стоимости состояния:

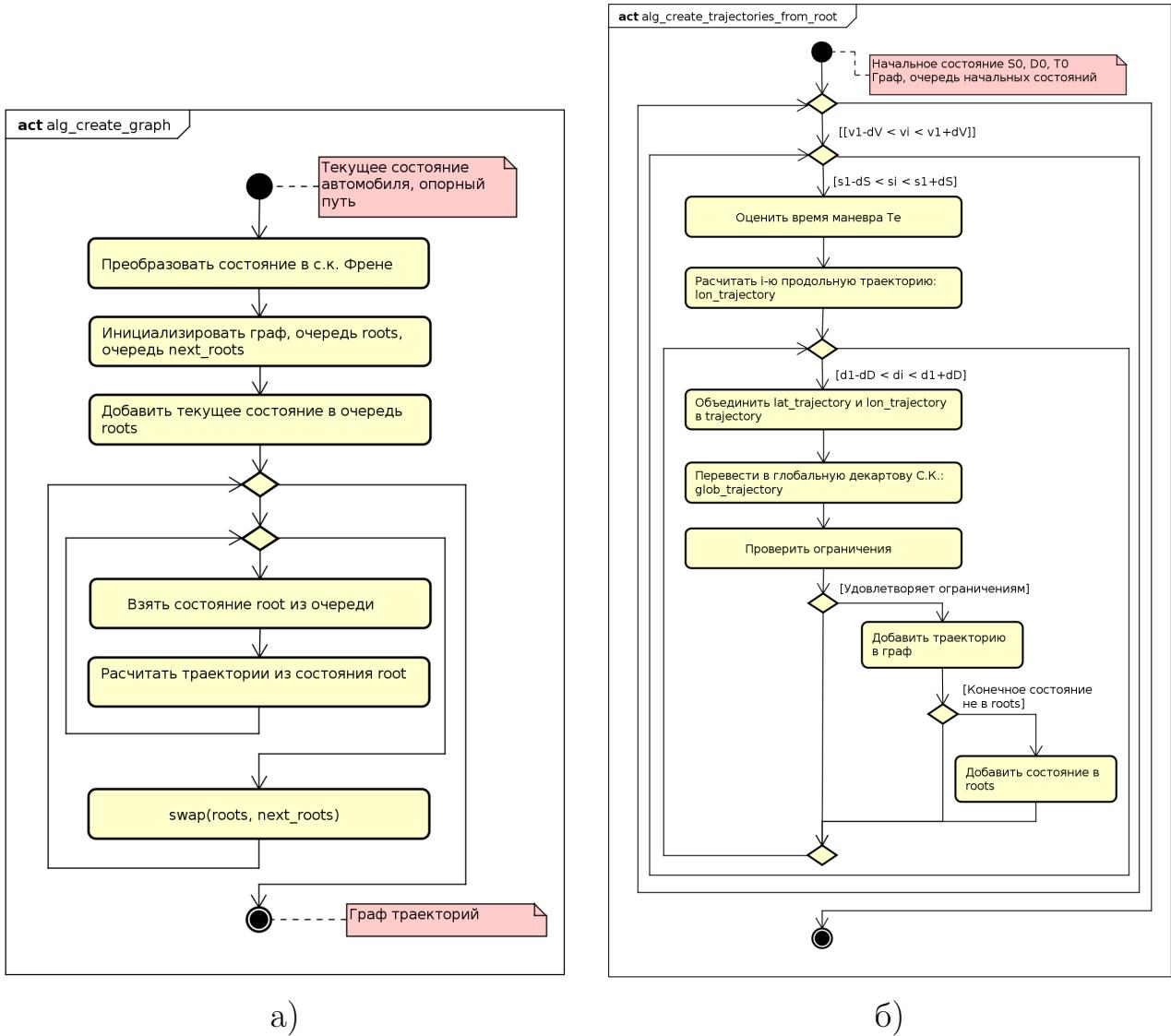
$$C_{move} = K_{lon} K_{sj} \int_0^T \dot{\ddot{s}}(t)^2 dt + K_{lat} K_{dj} \int_0^T \dot{\ddot{d}}(t)^2 dt \quad (26)$$

$$C_{state} = K_{lon} \left[K_s (s(T) - S_1)^2 + K_v (\dot{s}(T) - \dot{S}_1)^2 + K_{st} T \right] + \\ K_{lat} \left[K_{dd}(T)^2 + K_{dt} T \right] \quad (27)$$

Алгоритм планирования движения состоит из трех стадий: на первой осуществляется построение графа состояний с расчетом траекторий, являющихся ребрами, на второй осуществляется нахождением минимальной стоимости движения (т.е. учитывается только резкость маневров) до каждого конечного состояния с помощью алгоритма Дейкстры, на третьей осуществляется выбор оптимального конечного состояния с учетом стоимости движения и его удаленности от цели.

Блок-схема формирования набора траекторий приведена на рисунке 23.

На вход подаются состояние автомобиля, полученное от системы SLAM, карта препятствий и локальная цель. Вначале происходит преобразование начального состояния и целевого состояния в подвижную систему координат Френе, заданную опорной траекторией. Инициализируются пустой граф и две пустых очереди для хранения новых состояний, из которых для



а)

б)

Рисунок 23 – а) блок-схема алгоритма построения графа, б) блок схема шага расчета набора траекторий из заданного начального состояния.

которых осуществляется формирование траекторий. В качестве первого начального состояния в очередь помещается текущее состояние автомобиля в системе координат Френе. Алгоритм работает до тех пор, пока не будет построено необходимое количество слоев графа. При заполнении каждого слоя из очереди извлекается очередное состояние и осуществляется формирование набора траекторий, начинающихся из этого состояний. Конечные состояния этих траекторий помещаются в очередь, чтобы быть использованы для формирования нового слоя. В алгоритме используется две очереди состояний, которые переключаются после заполнения каждого слоя. Это сделано потому что в случае применения одной очереди, она бы постоянно пополнялась в процессе расчета траекторий. В данном случае, когда одна очередь становится

ся пустой, это означает, что из всех состояний текущего слоя были построены траектории, и происходит переключение на другую очередь, которая хранит начальные состояния для другого слоя.

Следующим этапом алгоритма планирования траектории является нахождение кратчайших путей до всех вершин с помощью алгоритма Дейкстры. Блок схема алгоритма изображена на рисунке 24.

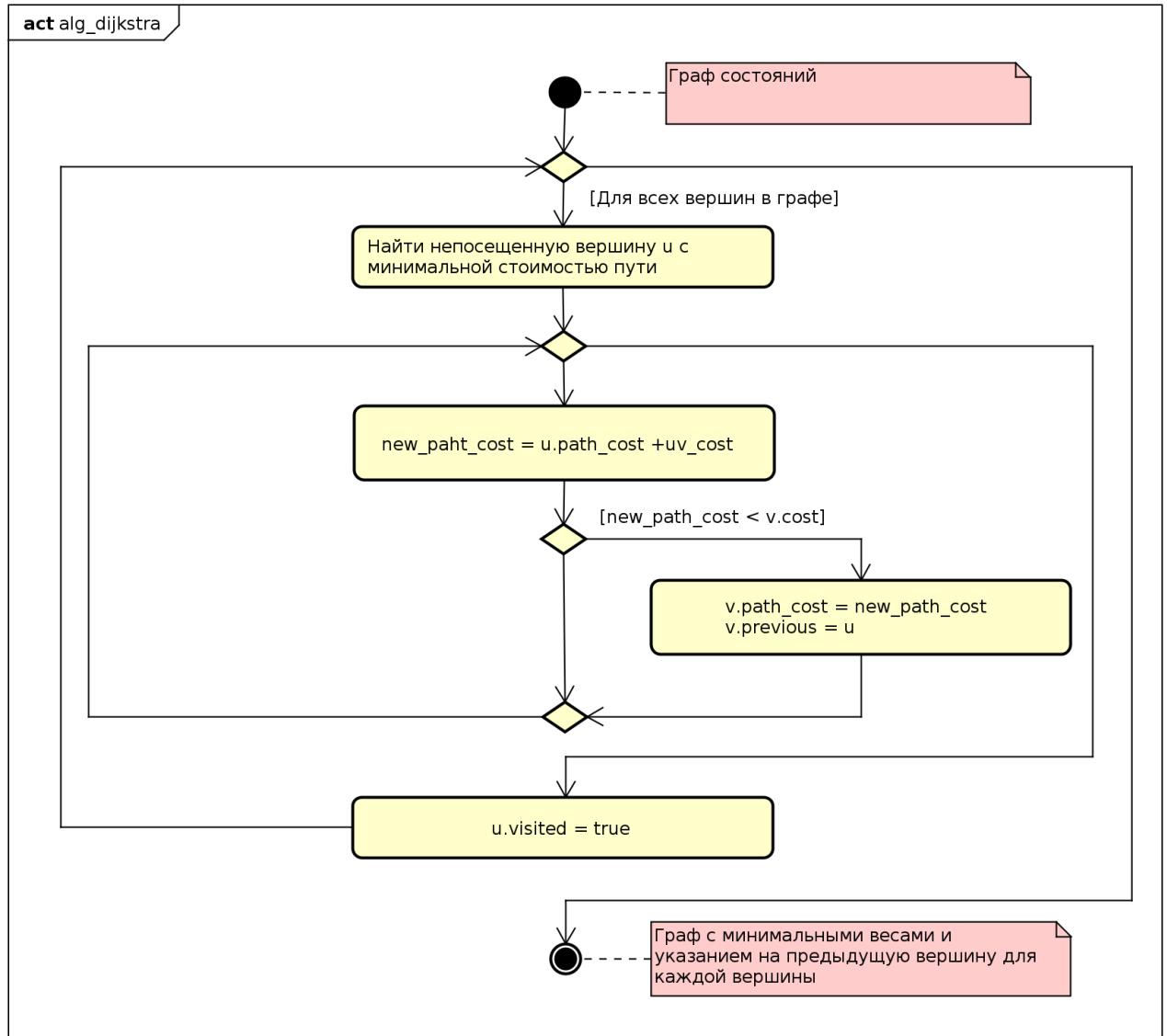


Рисунок 24 – Алгоритм Дейкстры

Вначале алгоритма стоимость пути до всех путей, кроме начальной, инициализируется равной бесконечности. Все вершины отмечаются, как непосещенные. Алгоритм Дейкстры состоит из количества итераций, равному количеству вершин в графе. На каждой итерации алгоритма выбирается вершина u с минимальной стоимостью пути из начальной до нее. Это может быть сделано простым перебором за $O(N)$, использование различных струк-

тур данных может сократить асимптотическую оценку, например, использование двоичной кучи позволяет находить вершину за $O(\log N)$. Более того, за счет того, что граф имеет регулярную структуру и проходится слой за слоем, можно ограничить диапазон поиска вершины с минимальной стоимостью ути. Затем перебираются все вершины v , связанные ребром с u . Рассчитывается, какая стоимость пути будет до вершин v , если двигаться к ней через вершину u : $\text{new_path_cost} = u.\text{path_cost} + uv.\text{edge_cost}$, и если эта стоимость меньше, чем та, которая сохранена в вершине v , значение стоимости пути до вершины v обновляется. Для того, чтобы в дальнейшем можно было восстановить кратчайший маршрут, а не только получить его стоимость, для вершины v обновляется предыдущая вершина кратчайшего маршрута — вершина u . После обработки всех связанных вершин, вершина u отмечается как посещенная, и после этого алгоритм переходит к следующей итерации.

Финальным этапом алгоритма является выбор оптимального конечного состояния, которое обладает минимальным суммарным весом $C = C_{\text{move}} + C_{\text{state}}$, где C_{move} получен в результате алгоритма Дейкстры, а C_{move} для каждой вершины рассчитан при формировании графа.

Пример формирования набора траекторий представлен на рисунке 25. Для наглядности осуществлялось варьирование только поперечных конечных состояний.

2.4 Выводы по главе

Был разработан алгоритм построения программной траектории движения для наземного транспортного средства. Алгоритм основан на алгоритме, предложенном командой "Junior" DARPA Urban Challenge [**darpa_junior**] и осуществляет нахождение траектории движения в форме полиномов пятого порядка. Отличие разработанного алгоритма от ближайшего аналога заключается в том, что он был расширен путем добавления возможности планирования на несколько шагов вперед. Это сделано путем составления графа возможных состояний автомобиля, ребра которого являются достижимыми траекториями в форме полиномов пятого порядка, и последующем нахождении пути с наименьшей стоимостью с использованием алгоритма Дейкстры.

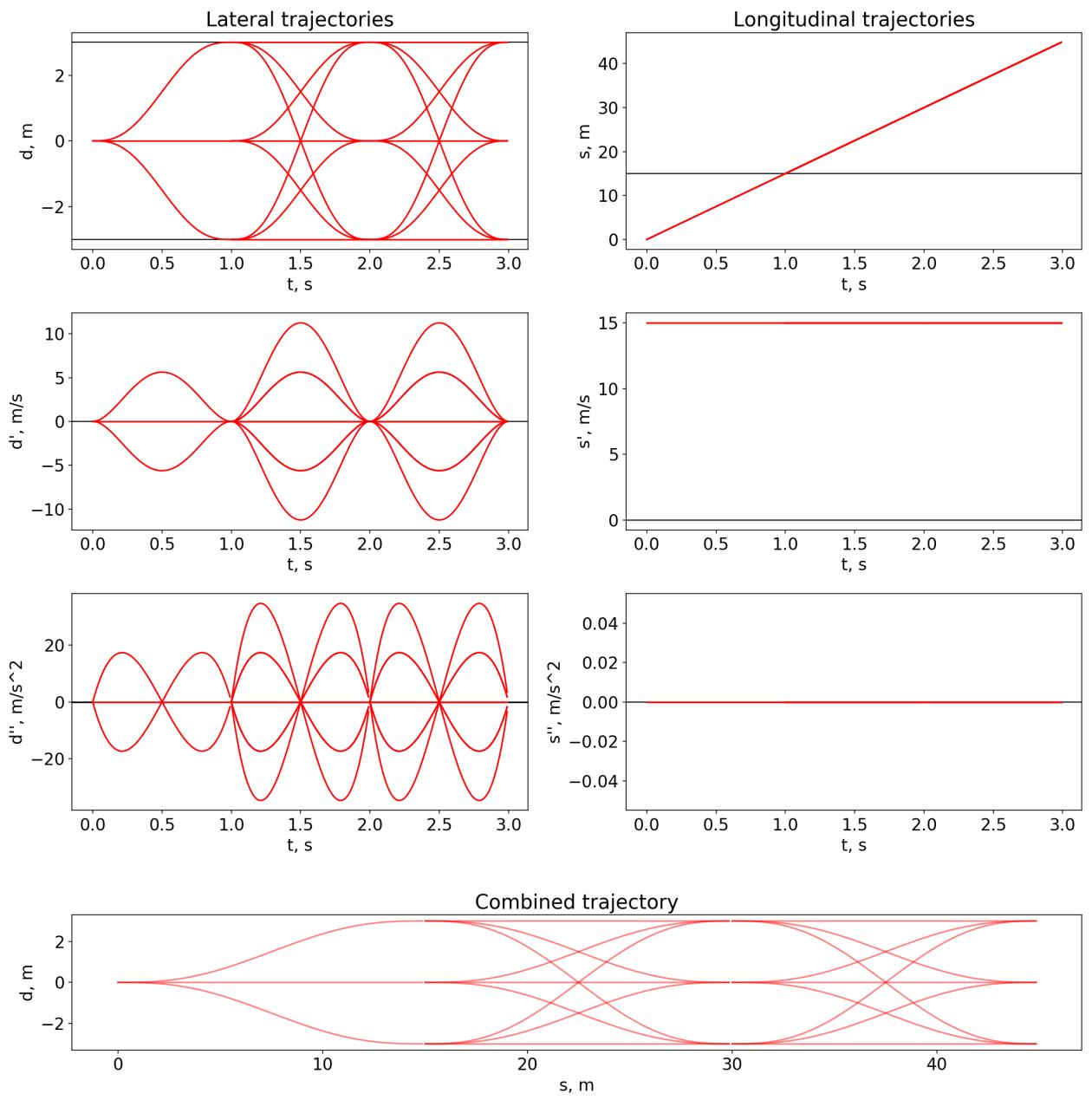


Рисунок 25 – Пример формирования набора траекторий

3 РЕАЛИЗАЦИЯ СИСТЕМЫ УПРАВЛЕНИЯ ДВИЖЕНИЕМ БЕСПИЛОТНОГО АВТОМОБИЛЯ

В этой главе рассматривается реализация системы управления движением беспилотного автомобиля на основе разработанного в предыдущей главе алгоритма, а также создание модели автомобиля для тестирования и отработки системы.

3.1 Использование ROS в качестве основы для системы управления

Разработка комплексного программного обеспечения, которым является система управления беспилотным автомобилем — сложная и объемная задача. Система управления состоит из ряда модулей, которые должны взаимодействовать друг с другом. Также требуется обеспечить работу с внешними устройствами, такими как стереокамера и LiDAR. Система управления должна состоять из следующих модулей:

- модуль получения видео со стереокамеры,
- модуль получения облака точек с LiDAR,
- модуль, реализующий алгоритм SLAM для определения положения и ориентации автомобиля в пространстве с помощью стереокамеры,
- модуль планирования движения,
- модуль движения по программной траектории,
- модуль управления двигателями,
- модуль пользовательского интерфейса для управления, визуализации и отладки системы.

Как видно, система состоит из большого числа модулей, для части из которых (работа с камерой и LiDAR, алгоритм SLAM) существуют готовые реализации. Необходимо реализовать модули достаточно независимыми, чтобы обеспечить легкость тестирования и возможность дальнейшего расширения и модернизации системы, что требует реализации способа взаимодействия между модулями.

Эта проблема широко распространена в сфере робототехники, и су-

ществует ряд решений, облегчающих разработку. Одним из таких решений является Robot Operation System (ROS) [[ros](#)].

Robot Operating System — это гибкий фреймворк для написания программного обеспечения для роботов. Он включает в себя набор инструментов, библиотек и соглашений, направленных на упрощение создания сложных систем управления для широкого класса робототехнических платформ. ROS является одним из самых широко распространенных фреймворком для разработки программного обеспечения для робототехнических систем.

Использование ROS обладает рядом преимуществ, таких как:

- поддерживает модульную и распределенную архитектуру приложений;
- обладает готовыми удобными механизмами взаимодействия между модулями;
- имеет большое количество утилит и инструментов, таких как программы с графическим интерфейсом, для отображения множества данных (изображения, облака точек, траектории и т.д.), логгер, позволяющий записывать данные, передаваемые между модулями, и потом воспроизводить их для дальнейшего изучения и отладки, визуализатор модулей, из которых состоит система, система сборки, облегчающая компиляцию и многое другое;
- множество готовых библиотек для управления, компьютерного зрения и визуализации;
- подробная документация;
- обширное сообщество разработчиков.

В основе ROS лежит концепция узла или ноды (node). Нода — это базовая единица построения ПО в ROS, программный модуль, который выполняет определенное действие, например, работает с LiDAR и предоставляет облака точек, и представляет собой отдельный процесс. Ноды взаимодействуют с помощью IPC (inter-process communication) механизмов, предоставляемых ROS: топиков (ROS topic) и сервисов (ROS service). Ноды ROS могут быть реализованы на различных языках программирования. Официально поддерживаются C++, Python 2.7 и Common Lisp, для этих языков реализованы клиентские библиотеки, позволяющие получить доступ к возможностям ROS. Существуют клиентские библиотеки для других языков,

тиакх как C#, Java, JavaScript, Ruby, Lua, Go, но большинство из них не обновляются и не поддерживают современные версии ROS. Помимо этого, существует пакет ROS "rosbridge_suite" предоставляющий интерфейс ROS в виде JSON-API, что позволяет использовать ROS с любым языком, в котором можно работать с сетью и JSON.

Топики реализуют механизм "издатель-подписчик". Топик является именованной типизированной FIFO очередью, в которую ноды могут асинхронно записывать данные и считывать данные. Ноды, которые записывают данные в топик называются издатель (publisher), а ноды, которыечитывают данные из ноды, называются подписчик (subscriber). Несколько нод могут одновременно писать и читать данные из одного топика.

Данные представляются в виде сообщений (messages). Сообщение представляет собой структуру данных, которая может состоять из примитивных типов данных, таких как целые числа или строки, и других типов данных, например, основанных на других сообщениях. Для описание сообщений в ROS применяются файлы специального формата, в которых перечисляются все тип и название всех полей сообщения. На основе этих файлов сборочная система ROS осуществляет генерацию кода для представления сообщения в поддерживаемых языках программирования. Так, например, для C++ и Python создаются классы с соответствующими полями и внутренний вспомогательный код для сериализации сообщений. ROS имеет большое количество встроенных стандартных типов сообщений, типичных для робототехники, такие как изображение (sensors_msgs/Image), облака точек (sensors_msgs/PointCloud2), положение и кватернион ориентации (geometry_msgs/Pose), путь, как набор положений (nav_msgs/Path), карта препятствий в виде сетки (nav_msgs/OccupancyGrid) и многие другие.

Преимуществом такого механизма является то, что отдельные ноды могут быть полностью независимы от других и не иметь никакой информации о них. Например, для ноды, занимающейся обработкой облака точек, совершенно не важно, получены ли данные от LiDAR, камеры глубины, такой как Kinect, от стереокамеры или даже воспроизводятся из файла — данные приходят в одинаковом стандартном формате.

Другим способом взаимодействия между нодами в ROS являются сервисы, реализующие механизм "запрос-ответ". В данном случае одна нода вы-

ступает в роли сервиса, а другие — в роли клиентов. Сервисы, в отличие от топиков, являются синхронными, т.е. нода, обратившаяся к сервису, ожидает ответа от него. Для описания сервисов также существует определенный формат файла, в котором описываются типы и имена аргументов, передаваемые клиентом сервису, и формат ответа от сервиса. Таким образом, сервис реализует семантику удаленного вызова процедуры (Remote Procedure Call).

ROS обладает центральным узлом — мастером (ROS Master), который отвечает за соединение между собой всех узлов ROS. Мастер хранит информацию обо всех подписках, публикациях, сервисах и предоставляет эту информацию нодам. Взаимодействие между нодами и мастером осуществляется по протоколу XML/RPC, дальнейшее взаимодействие нод между собой может осуществляться по ряду протоколов, чаще всего применяется TCP.

На рисунке 26 приведена иллюстрация механизма установления соединения между двумя нодами при использовании топиков.

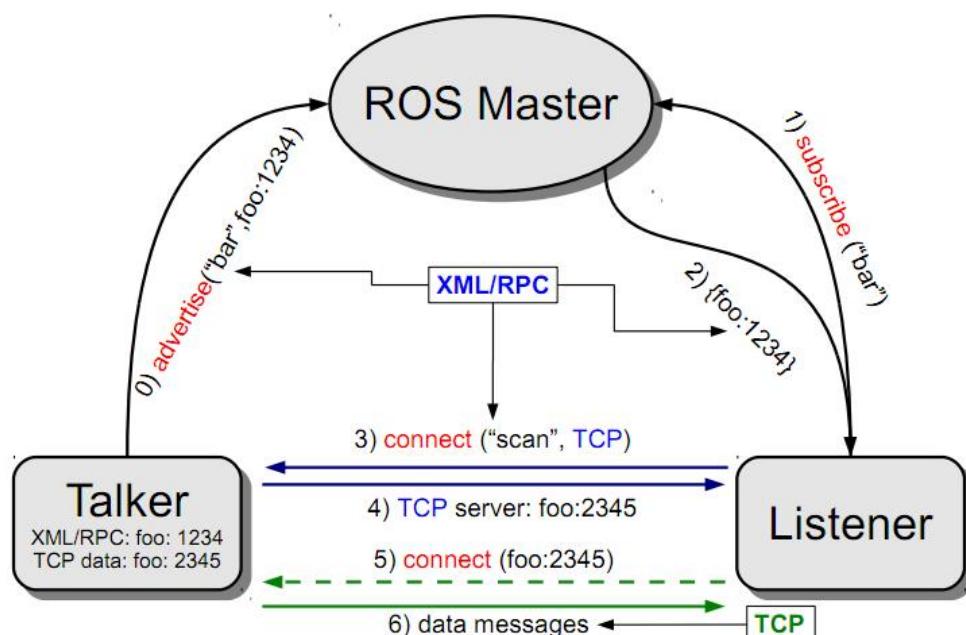


Рисунок 26 – Механизм работы топиков ROS

Нода, выступающая в роли издателя уведомляет об этом мастера, вызывая метод "advertise" с использованием протокола XML/RPC, сообщая ему имя топика и тип сообщения. Нода, выступающая в качестве подписчика, отправляет запрос "subscribe" мастеру с использованием протокола XML/RPC, сообщая имя топика, на который осуществляется подписка. В ответ мастер сообщает данные для подключения к ноде-издателю. После этого нода-

слушатель отправляет запрос "connect" ноде-издателю с использованием протокола XML/RPC. В ответ нода-издателю сообщает данные (адрес и порт) для установления TCP-соединения. После этого нода-подписчик осуществляет TCP-подключение к ноде-издателю и после этого нода-издателю может осуществлять передачу данных.

Механизм работы сервисов приведен на рисунке 27.

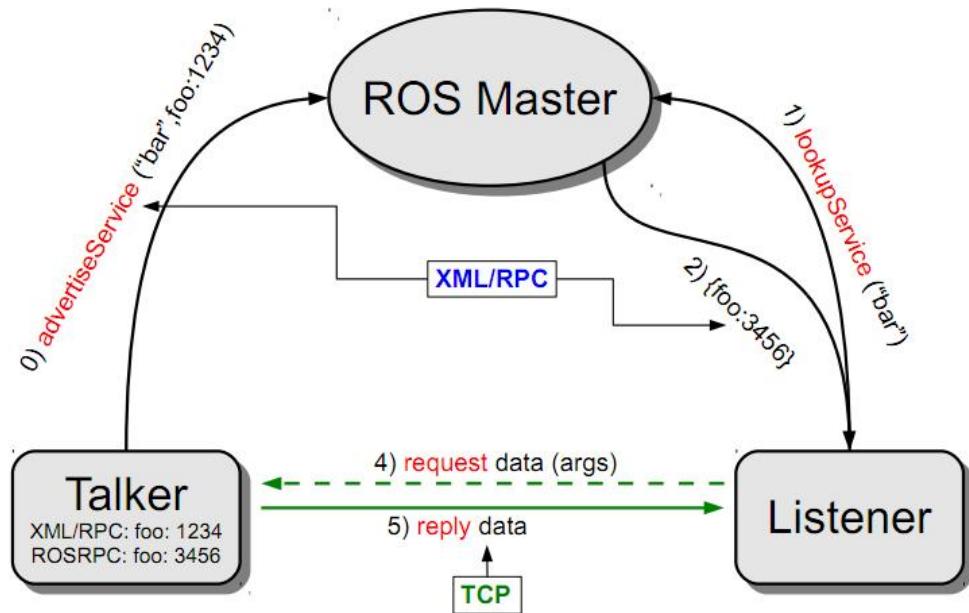


Рисунок 27 – Механизм работы топиков ROS

В данном случае нода, выступающая в роли сервиса, уведомляет об этом мастера, вызывая метод "advertiseService" с использованием протокола XML/RPC. Нода, использующая сервис, вначале запрашивает у мастера данные для вызова сервиса, вызывая метод "lookupService" с использованием протокола XML/RPC. Мастер сообщает данные для подключения (адрес и порт). После этого, когда требуется вызвать сервис, нода устанавливает TCP-подключение, отправляет запрос и принимает ответ от ноды-сервиса.

Существует два режима работы сервисов: с установлением соединения на каждый запрос (по-умолчанию), и с постоянным соединением. Первый режим более надежен, т.к. позволит работать даже в случае перезапуска сервиса, а второй режим обеспечивает меньшую латентность вызова сервиса. Результаты измерения латентности вызова сервиса в режиме повторного подключения и режиме постоянного подключения в пределах localhost представлены на рисунке 28 и в таблице 2. Из результатов измерения видно, что посто-

янное подключение обеспечивает существенно меньшую латентность вызова и, что не менее важно, меньший джиттер (разброс времени вызова), что особенно важно для систем реального времени.

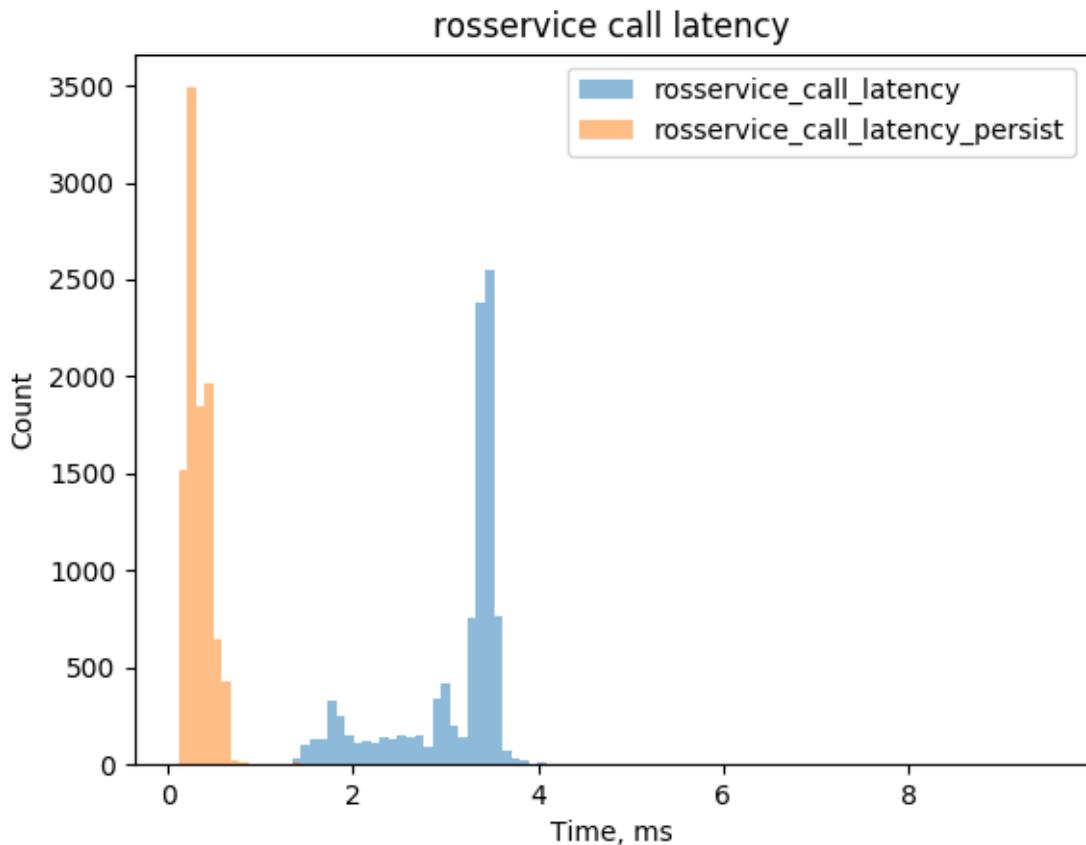


Рисунок 28 – Механизм работы топиков ROS

Таблица 2 – Сравнение латентности вызова сервисов ROS в разных режимах

	Повторное подключение, мс	Постоянное подключение, мс
Минимальное	1.2	0.1
Среднее	3.1	0.4
Максимальное	9.5	1.1

3.2 Постройка мобильной платформы

Для испытаний и отладки системы управления движением беспилотного автомобиля была построена небольшая модель автомобиля на базе шестиколесной платформы. Внешний вид модели представлен на рисунке 29.

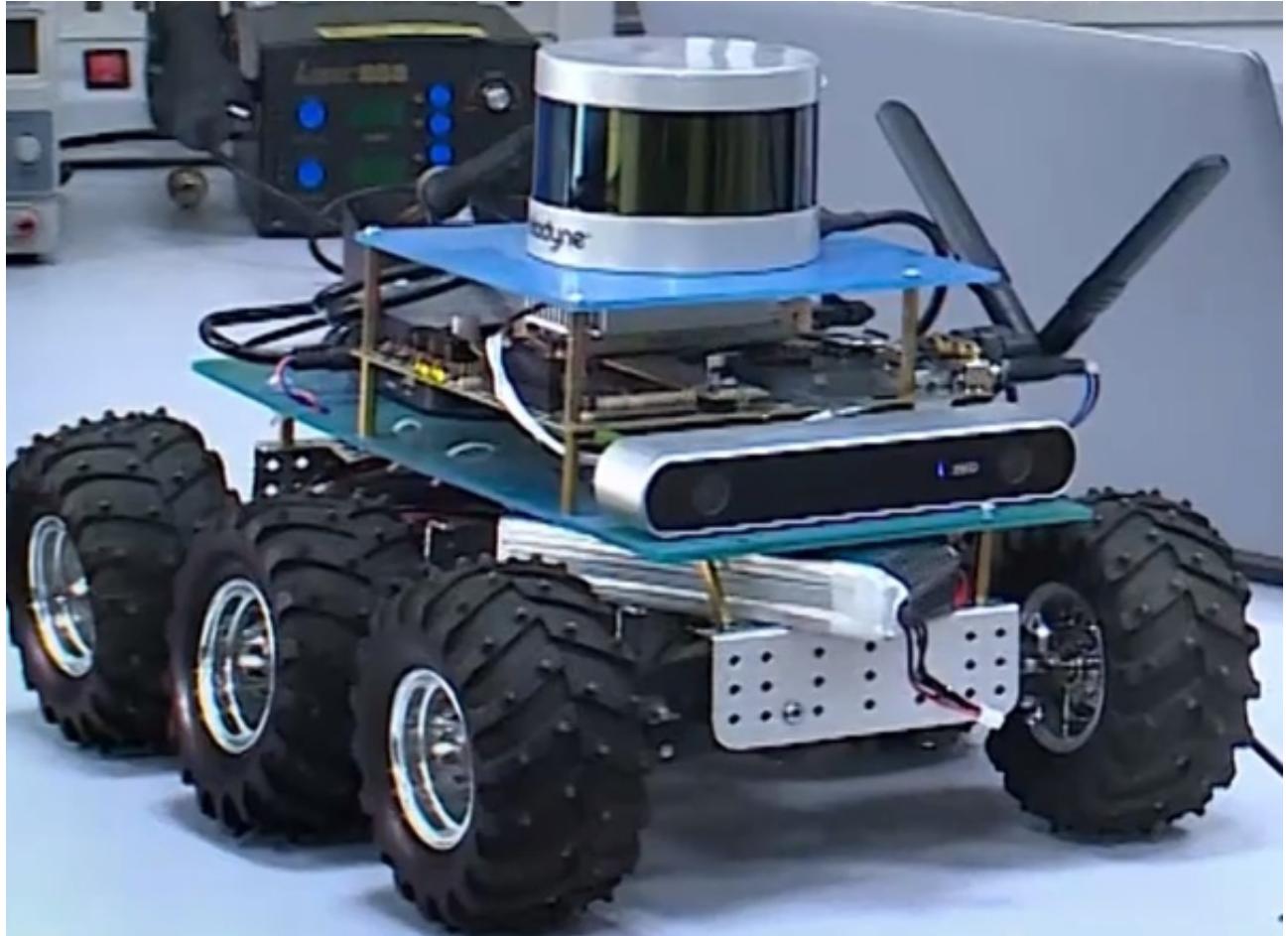


Рисунок 29 – Внешний вид модели

Модель построена на базе шестиколесной платформы Dagu Wild Thumper 6WD [[jetson_car_6wd](#)], оснащенной шестью двигателями постоянного тока. Данная платформа кинематическую схему, которая отличается от кинематической схемы обычного автомобиля (схемы Аккермана), управляетя путем изменения относительной скорости вращения колес левого и правого бортов, и, следовательно, способна на разворот на месте. Тем не менее, на начальном этапе испытаний системы управления движением беспилотного автомобиля она позволяет провести ряд экспериментов и частично отладить систему управления. Эта платформа была выбрана по причине того, что она была доступна на кафедре ЭВМиС Волгоградского Государственного Университета. Двигатели левого и правого борта соединены параллельно (по три двигателя в параллель) и подключены к управляются двумя драйверами VNH3SP30 [[jetson_car_driver](#)].

Для определения положения и ориентации автомобиля применяется алгоритм SLAM и стереокамера ZED Camera [[jetson_car_zed](#)] от компании

Stereolabs. Камера обладает качественной оптикой с широким углом обзора в 90° и 60° по горизонтали и вертикали соответственно и максимальным суммарным разрешение двух камер 4416x1242. В комплекте к камере поставляется программное обеспечение, реализующее ряд возможностей, в том числе алгоритм SLAM, позволяющий отслеживать положение и ориентацию камеры в пространстве. Производителем заявляется точность позиционирования в 1 мм и точность определения ориентации в 0.1°, но без уточнения условий, в которых получены такие результаты, потому что для алгоритмов SLAM характерно накопление ошибки при увеличении пройденного расстояния. Помимо этого точность SLAM зависит от окружающего пространства, наличия ключевых точек, которые могут быть детектированы, наличия подвижных объектов, которые могут ввести алгоритм в заблуждение. Специальные эксперименты по оценке точности ZED-камеры не проводились, но по результатам эксплуатации можно сделать вывод, что заявленные производителем характеристики не соблюдаются. Тем не менее, ZED-камера показала хорошую точность при небольших перемещениях (в пределах помещений), достаточную для проведения экспериментов с моделью беспилотного автомобиля.

Для определения препятствий применяется LiDAR Velodyne VLP-16. LiDAR позволяет получить трехмерное облако точек, представляющее окружающую обстановку, обладающее сравнительно большой детализацией, обнаруживая препятствия на расстояниях до ста метров. Качественное облако точек существенно облегчает задачу определения препятствий, что позволяет сконцентрировать усилия на основной задаче данной работы — разработке системы планирования движения. Т.к. LiDAR уже был приобретен университетом для проекта беспилотного автомобиля, его использования выглядит оправданным.

В качестве бортового компьютера используется встраиваемая система NVidia Jetson TX2. Это встраиваемая система на базе процессора с архитектурой ARM и встроенного графического ускорителя с архитектурой NVidia Pascal, предназначенная для применения в задачах компьютерного зрения и нейронных сетей. Эта встраиваемая система обладает малым энергопотреблением, что важно для мобильной платформы с питанием от аккумуляторных батарей, по сравнению с системами на базе x86 процессоров, но, в то же время, достаточно высокой производительностью по сравнению с другими системами.

ми на базе ARM, такими как Raspberry Pi. Наличие производительного GPU позволяет использовать ресурсоемкие алгоритмы. В данном случае, GPU используется для работы алгоритма SLAM ZED-камеры, который реализован с использованием CUDA.

Питание мобильной платформы осуществляется от двух литий-полимерных аккумуляторов с напряжением 7.4 В (2S, две аккумуляторных ячейки) и 12 В (3S, три аккумуляторных ячейки). Два аккумулятора применяются для раздельного питания электроники (NVidia Jetson TX2 и LiDAR) и двигателей, чтобы исключить негативные влияния возможных импульсов напряжения или просадок напряжения на чувствительную электронику.

Для управления двигателями применяется микроконтроллер STM32F103C8T6, который получает команды управления от NVidia Jetson по интерфейсу UART. Отдельный микроконтроллер применяется по причине того, что NVidia Jetson не имеет аппаратного ШИМ.

3.3 Реализация подсистемы распознавание препятствий

Для тестирования алгоритма планирования движения необходимо определять препятствия, которые будут обходить моделью автомобиля. Детектирование препятствий осуществляется с помощью облака точек, получаемого от LiDAR, а карта препятствий представляется в виде OccupancyGrid — регулярной сетки, где каждая клетка может быть в одном из трех состояний: неизвестно, есть препятствие, нет препятствия (более точно, клетка в OccupancyGrid определяет вероятность нахождения препятствия в ней, но в данной работе вероятностный подход не применялся).

Первоначальный вариант алгоритма определения основывался на детектировании (сегментации) плоскости поверхности, что является распространенным подходом для решения этой задачи. Самым простым способом сегментации плоскости является алгоритм Random Sample Consensus (RANSAC) [ransac]. Этот алгоритм позволяет осуществлять регрессию данных, в которых присутствует большое количество выбросов. В результате алгоритм определяет находит параметры модели, а также разделяет входные данные на удовлетворяющие модели (inliers) и выбросы (outliers).

Алгоритм работает следующим образом. На каждой итерации из об-

щего набора экспериментальных данных X выбирается M случайных точек X_1 . Вычисляются параметры θ модели. Для задачи сегментации плоскости применяется линейная регрессия. Затем, проверяется, какие точки общего набора данных X отклоняются от модели не более, чем на p . Отклоняющиеся точки помечаются как outliers, а соответствующие — как inliers. Если количество инлаеров больше, чем максимальное, то происходит обновление максимального числа инлаеров и коэффициентов модели. Алгоритм повторяется N итераций. Благодаря случайному выбору точек алгоритм RANSAC более устойчивый к выбросам, чем обычная линейная регрессия. В работе использовалась реализация алгоритма RANSAC из библиотеки Point Cloud Library.

После сегментации облака точек и определения точек, принадлежащих плоскости и принадлежащих препятствиям, облако точек преобразуется в OccupancyGrid. Для этого создается OccupancyGrid с размером, соответствующим размеру облака точек, и заданным размером сетки, например, 0.25 м. Зная координату каждой точки, координату начала координат OccupancyGrid и размер клетки, можно определить индекс клетки, к которой принадлежит точка:

$$p_i = \lfloor \frac{p_x - x_0}{cell_size} \rfloor \quad (28)$$

$$p_j = \lfloor \frac{p_y - y_0}{cell_size} \rfloor \quad (29)$$

Клетка считается препятствием, если количество точек, отмеченных как препятствие, попавших в нее, превышает некое установленное значение.

Velodyne VLP-16 обладает шестнадцатью лучами, кроме того, мобильная платформа обладает никой высотой, поэтому количество точек в облаке точек, принадлежащих поверхности, мало. Это приводит к тому, если что считать допустимой для движения областью только области, сегментированные как плоскость, то большая часть пространства не будет отмечена как допустимая. Более эффективным способом будет считать все клетки с препятствиями как препятствия, а все прочие клетки — как свободные, исключая те, которые находятся "в тени". Для этого применяется метод трассировки луча (ray casting) в 2D. От LiDAR до первой клетки препятствия все клетки

считываются как свободные, затем идет клетка препятствия, а далее все клетки помечаются как неизвестные. Т.к. OccupancyGrid представляет собой сетку, для трассировки луча применяется алгоритм Брезенхема для растеризации линии, широко применяемый в компьютерной графике.

Помимо трассировки линий, вокруг каждой клетки препятствия расстирается окружность заданного радиуса, для того, чтобы обеспечить безопасную зону вокруг препятствий. Благодаря этому при определении пересечения автомобиля с препятствием можно считать автомобиль точечным. Данный метод не учитывает ориентацию автомобиля. Т.к. автомобиль имеет вытянутую форму, приходится выставлять безопасный радиус по самому большому габариту модели, т.е. равному длине. Это приводит к неэффективному использованию пространства, особенно в узких проходах. Результат работы алгоритма приведен на рисунке 30.

Система детектирования препятствий была реализована в виде ноды ROS, которая подписана на топики облака точек и текущее положение модели и публикует nav_msgs/OccupancyGrid.

3.4 Реализация подсистемы планирования траектории

Система планирования траектории реализована в виде ноды ROS. Нода подписывается на два топика: /car_state и /next_target. Топик /car_state имеет тип нестандартный сообщений car_msgs/CarState и представляет текущее состояние автомобиля: положение, ориентацию, линейную скорость, линейное ускорение. Топик /next_target имеет нестандартный тип car_msgs/MotionPlanningTarget и представляет цель для планирования движения: опорную траекторию, целевое положение, которое должно лежать на опорной траектории и целевую скорость. Нода публикует найденную оптимальную траекторию в топик /local_path, имеющий нестандартный тип сообщения car_msgs/CarMotion и содержит найденные путь, профиль скорости и ускорения в глобальной декартовой системе координат, название которой указывается стандартным для ROS способом в поле frame_id сообщения.

Прототип реализован на языке Python 2.7. Диаграмма классов системы планирования движения представлена на рисунке 31.

Главным Классом является класс MotionPlanner, который реа-

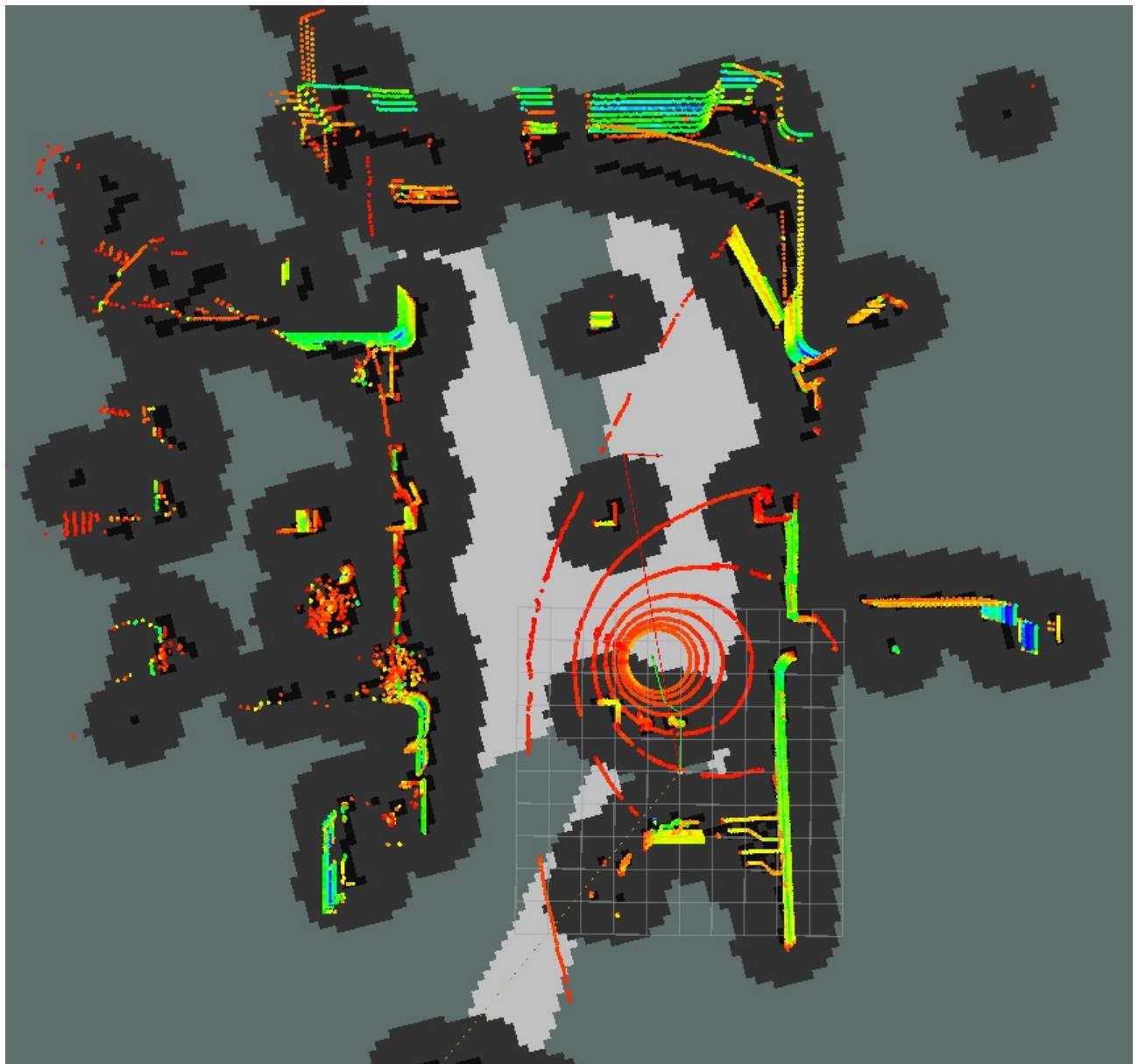


Рисунок 30 – Результат работы алгоритма детектирования препятствий

лизует алгоритм планирования траектории и используется из ноды motion_planner_node. Реализованы следующие вспомогательные классы: Quintic, Frenet, Trajectory1D, Trajectory2D, Vertex. Класс Quintic отвечает за расчет коэффициентов полиномов пятого порядка и их интерполяцию. Класс Frenet представляет систему координат Френе и осуществляет преобразование координат в эту систему и обратно в глобальную декартову систему координат. Классы Trajectory1D и Trajectory2D представляют одномерную (продольную либо поперечную) и двумерную (в системе координат Френе или в глобальной системе координат) траектории соответственно. Эти класс хранят траекторию, первую и вторую производную и рассчитанный вес этой траектории. Класс Vertex представляет вершину графа и применяется в алгоритме

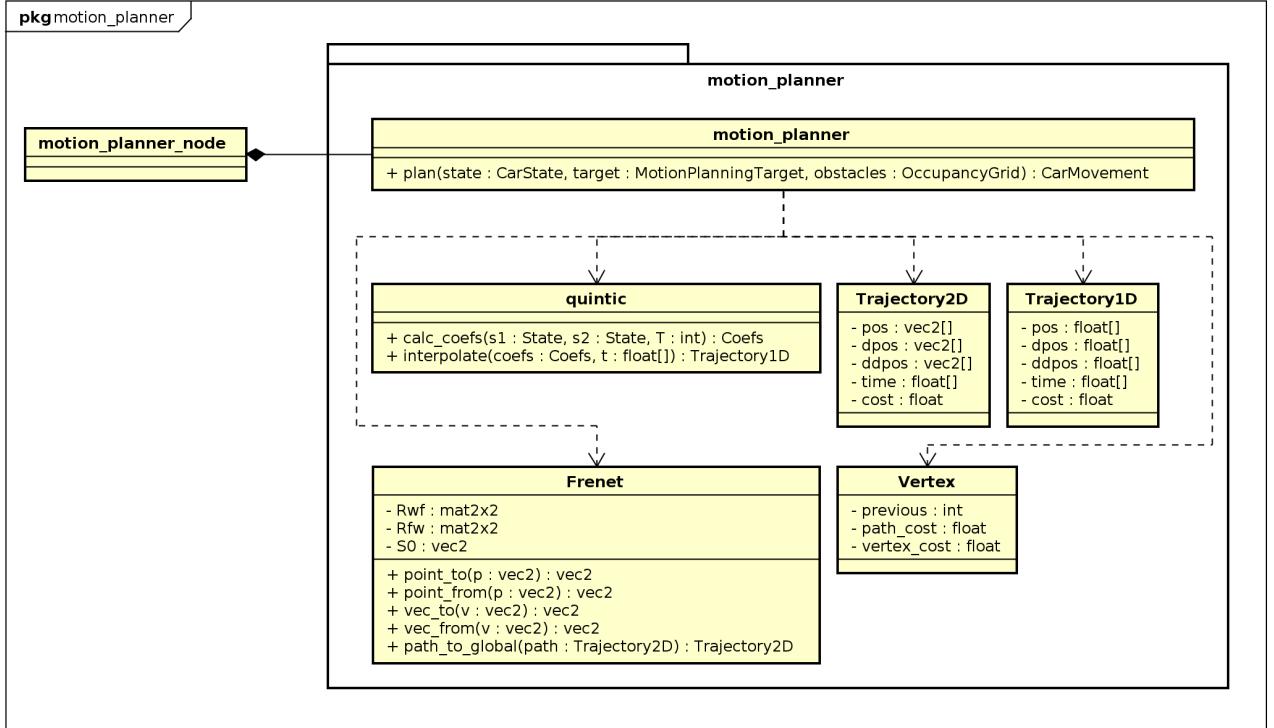


Рисунок 31 – Интерфейс ноды планирования движения

Дейкстры.

Алгоритм планирования движения требует определения текущей скорости автомобиля, в то время как с помощью SLAM алгоритма можно получить только текущие координаты. Для получения скорости была реализована нода `car_state_publisher`, которая осуществляет численное дифференцирование получаемых от ZED-камеры координат для определения скорости. Эксперименты показали, что результаты дифференцирования координат зашумлены. Для решения этой проблемы был применен фильтр скользящей медианы. Пример исходных данных и результат фильтрации приведен на рисунке 32. На правом изображении приведены графики положения камеры по x - и y -координатам. На левом изображении приведены графики скоростей, полученные путем дифференцирования положения, и отфильтрованные с помощью фильтра скользящей медианы.

3.5 Реализация подсистемы следования траектории

Оставшимся компонентом системы управления движением беспилотного автомобиля является подсистеме следования по траектории, т.е. регулятор с обратной связью. Назначение этой подсистемы заключается в том,

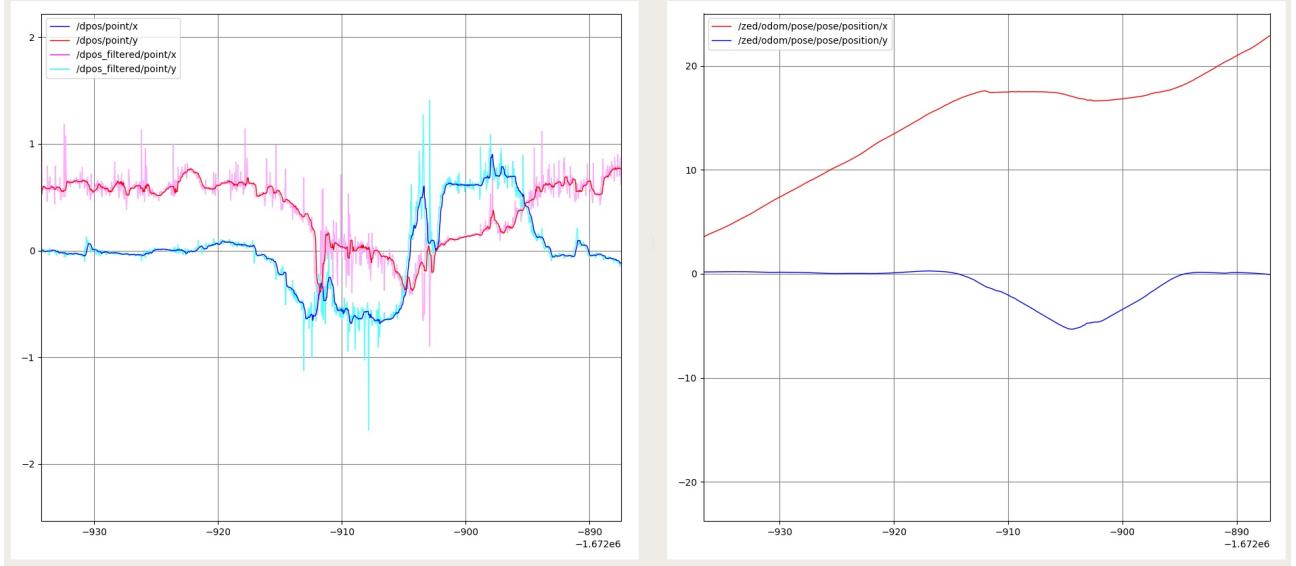


Рисунок 32 – Результаты расчета скорости по данным с ZED-камеры

чтобы осуществлять выполнение траекторий, которые были сгенерированы подсистемой планирования движения, формируя последовательность управляющих сигналов для актуаторов автомобиля. Чтобы избежать неизбежных отклонений от траектории, которые неизбежно возникнут в системе открытого цикла, применяется обратная связь, использующая текущее состояние автомобиля для коррекции последовательности команд. В данном случае, обратная связь осуществляется с помощью положения и ориентации автомобиля, получаемые от ZED-камеры.

В данной работе для рулевого управления применяется регулятор с обратной связью, основанный на модели МакАдама **TODO: ссылка**. Иллюстрация работы регулятора с обратной связью приведена на рисунке 33.

Принцип работы этого алгоритма заключается в том, чтобы найти вектор \vec{OV} , соединяющий текущее положение автомобиля O с траекторией и использовать угол между текущим вектором скорости \vec{v} и найденным вектором в пропорциональном (Π)-регуляторе, чтобы определить управляющие сигналы. Вектор \vec{OV} выбирается таким образом, чтобы он соединял текущее положение автомобиля с траекторией на неком удалении вперед от текущего положения автомобиля. Это позволяет принимать во внимание будущие изменения траектории и начинать маневр заранее.

Вектор \vec{OP} находится как точка пересечения окружности радиусом R и траектории. Траектория представлена набором опорных точек с малым расстоянием друг между другом. Это позволяет легко находить положение

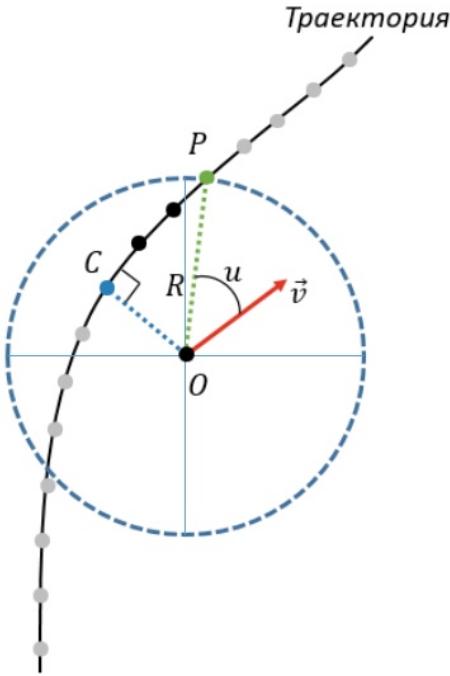


Рисунок 33 – Иллюстрация работы регулятора с обратной связью

точки P с точностью до расстояния между двумя последовательными точками траектории. Возможно было бы повысить точность определения точки P путем нахождения пересечения окружности с отрезком, образованном двумя ближайшими точками траектории, но это не требуется. Алгоритм работает следующим образом. Вначале находится точка C траектории, ближайшая к текущему положению автомобиля O . Затем от точки C по направлению движения траектории отсчитываются точки до тех пор, пока очередная точка не будет удалена от точки O на расстояние больше, чем R . Это и будет искомая точка P . Угол u вычисляется следующим образом:

$$\vec{v} \times \overrightarrow{OP} = |\vec{v}| |\overrightarrow{OP}| \sin u \quad (30)$$

$$\vec{v} \cdot \overrightarrow{OP} = |\vec{v}| |\overrightarrow{OP}| \cos u \quad (31)$$

(32)

$$\tan u = \frac{\vec{v} \times \overrightarrow{OP}}{\vec{v} \cdot \overrightarrow{OP}} = \frac{|\vec{v}| |\overrightarrow{OP}| \sin u}{|\vec{v}| |\overrightarrow{OP}| \cos u} = \frac{\sin u}{\cos u} \quad (33)$$

$$u = \text{atan2}(\vec{v} \times \overrightarrow{OP}, \vec{v} \cdot \overrightarrow{OP}) \quad (34)$$

Где функция *atan2* определена следующим образом:

$$\text{atan2}(y, x) = \begin{cases} \arctan\left(\frac{y}{x}\right), & \text{при } x > 0, \\ \arctan\left(\frac{y}{x}\right) + \pi, & \text{при } x < 0, y \geq 0, \\ \arctan\left(\frac{y}{x}\right) - \pi, & \text{при } x < 0, y < 0, \\ +\frac{\pi}{2}, & \text{при } x = 0, y > 0, \\ -\frac{\pi}{2}, & \text{при } x = 0, y < 0, \\ \text{не определено,} & \text{при } x = 0, y = 0 \end{cases} \quad (35)$$

Использование такого выражения позволит получить значение угла со знаком во всех тригонометрических четвертях. Знак угла используется для определения направления поворота (налево при $u > 0$ или направо при $u < 0$). Функция *atan2* реализована в стандартных математических библиотеках большинства языков программирования.

Т.к. мобильная платформа, на которой проводились эксперименты, обладает дифференциальным приводом, т.е. поворот осуществляется за счет изменения скорости вращения колес левого и правого борта друг относительно друга, то скорости вращения колес определяются на основе расчетного угла u с использованием пропорционального регулятора следующим образом:

$$left = speed + ku \quad (36)$$

$$right = speed - ku \quad (37)$$

Этот компонент реализован на языке Python 2.7 в виде ноды ROS *path_mover_node*. Нода подписывается на текущее состояние автомобиля, получаемое от ZED-камеры */zed/pose* и на локальную траекторию, сгенерированную планировщиком локального движения (*motion_planner_node*) *local_path*.

3.6 Выводы по главе

Были реализованы основные компоненты системы управления движением беспилотного автомобиля, которые были описаны во второй главе.

Структура системы управления движением основывается на фреймворке ROS и использует его как связующий слой для модулей встраиваемой системы управления.

Реализована подсистема планирования движения, согласно алгоритму, проектирование которого было описано во второй главе, которая является основным результатом данной работы. Подсистема планирования движения позволяет планировать траектории, позволяющие достичь локальной цели, избегая столкновения с препятствиями. Реализован прототип системы компьютерного зрения для обнаружения препятствия с помощью LiDAR, используемые для испытаний и отладки системы управления. Реализована подсистема удержания траектории в виде регулятора с обратной связью, позволяющая двигаться по траекториям, сгенерированным подсистемой планирования движения. Реализован компонент драйвера, позволяющий отправлять команды управления аппаратному обеспечению автомобиля.

Изготовлена мобильная колесная платформа, оснащенная датчиками системы компьютерного зрения, позволяющая производить испытания и отладку системы управления автомобилем.

4 ЭКСПЕРИМЕНТЫ И ОЦЕНКА РЕЗУЛЬТАТОВ РАБОТЫ

В данной главе рассматривается этап оценки работы подсистемы управления движением. Описанные проведенные эксперименты по проверке работы системы управления

4.1 Экспериментальное исследование подсистемы движения по траектории

Подсистема движения по траектории играет важную роль в работе системы управления движением автомобиля, т.к. точное следование траектории требуется, чтобы автомобиль двигался так, как было запланировано системой планирования движения, чтобы обеспечить безопасное и оптимальное движение.

Были произведены испытания подсистемы движения по траектории отдельно от системы планирования движения. На вход системы подавались искусственно сформированные траектории различной формы, осуществлялось движение модели автомобиля по этой траектории, и оценивалось реальная траектория с помощью визуального наблюдения, измерения расстояния между ключевыми точками траектории и анализируя траекторию движения, полученную с ZED-камеры. На рисунке 34. По результатам испытаний было установлено, что максимальное отклонение при движении по окружности радиус 1.5 м составило 50 мм.

4.2 Экспериментальное исследование подсистемы планирования локального движения

Подсистема планирования локального движения является одной из ключевых подсистем в системе управления беспилотным автомобилем. Был произведен ряд экспериментов с использованием мобильной колесной платформы для оценки работы алгоритмов.

Первый вариант подсистемы планирования движения основывался на алгоритме А, планируя геометрический путь в обход препятствия, когда оно

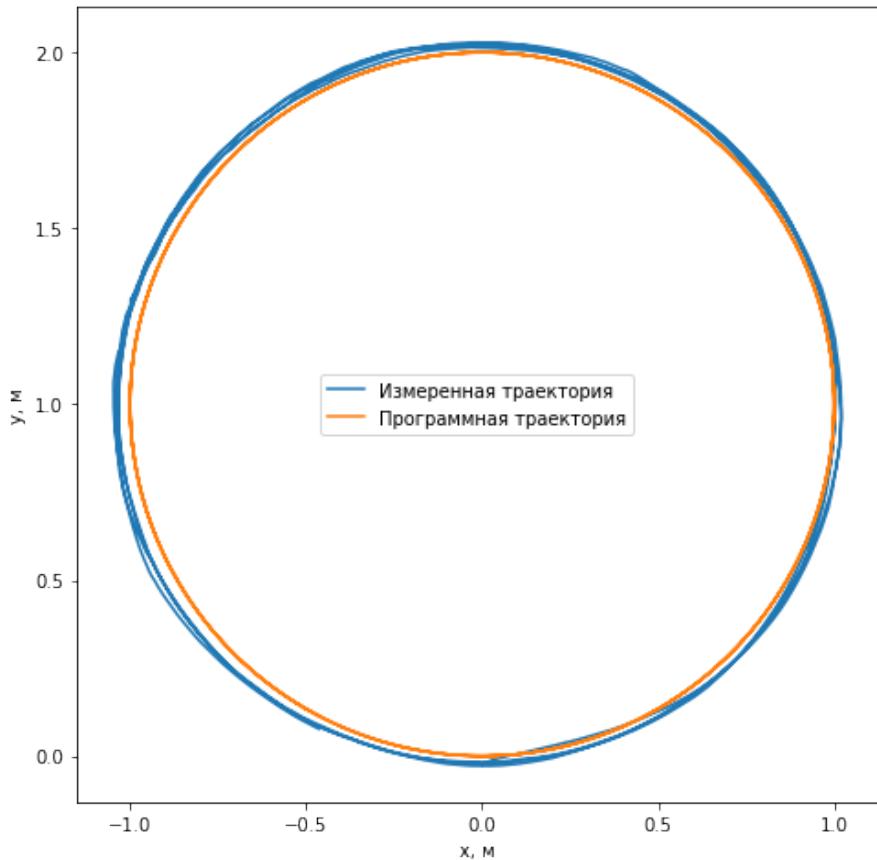


Рисунок 34 – Результаты эксперимента с движением по заданной траектории

пересекало опорную траекторию. Алгоритм был реализован и испытан на мобильной платформе. По результатам испытаний были выявлены дополнительные недостатки алгоритма. В связи с неидеальностью системы компьютерного зрения, при обновлении периодическом Occupancy Grid происходит небольшое "мерцание" т.е. те или иные пограничные ячейки карты могут менять свое состояние. Это приводит к тому, что алгоритм резко перестраивает траекторию движения. Особенно это было заметно, когда модель приближалась к симметричному препятствию. В таком случае, алгоритм мог прокладывать обходную траекторию то по левую сторону от препятствия, то по правую сторону от препятствия. Это приводило к тому, что модель не могла корректно осуществить маневр и врезалась в препятствие.

По результатам испытаний было выявлено, что рассмотренный алгоритм не подходит для локального планирования траекторий движения беспилотного автомобиля.

Следующий вариант подсистемы планирования движения движения использует планирование траектории в форме полиномов пятого порядка и

учитывает текущую ориентацию и скорость модели, чтобы планировать движения по гладким траекториям. Результаты испытаний этого алгоритма показали, что алгоритм способен осуществлять движение по опорной траектории, осуществляя маневры для обезвреживания препятствий с последующим возвращением на траекторию. Результат работы алгоритма приведен на рисунке 35.

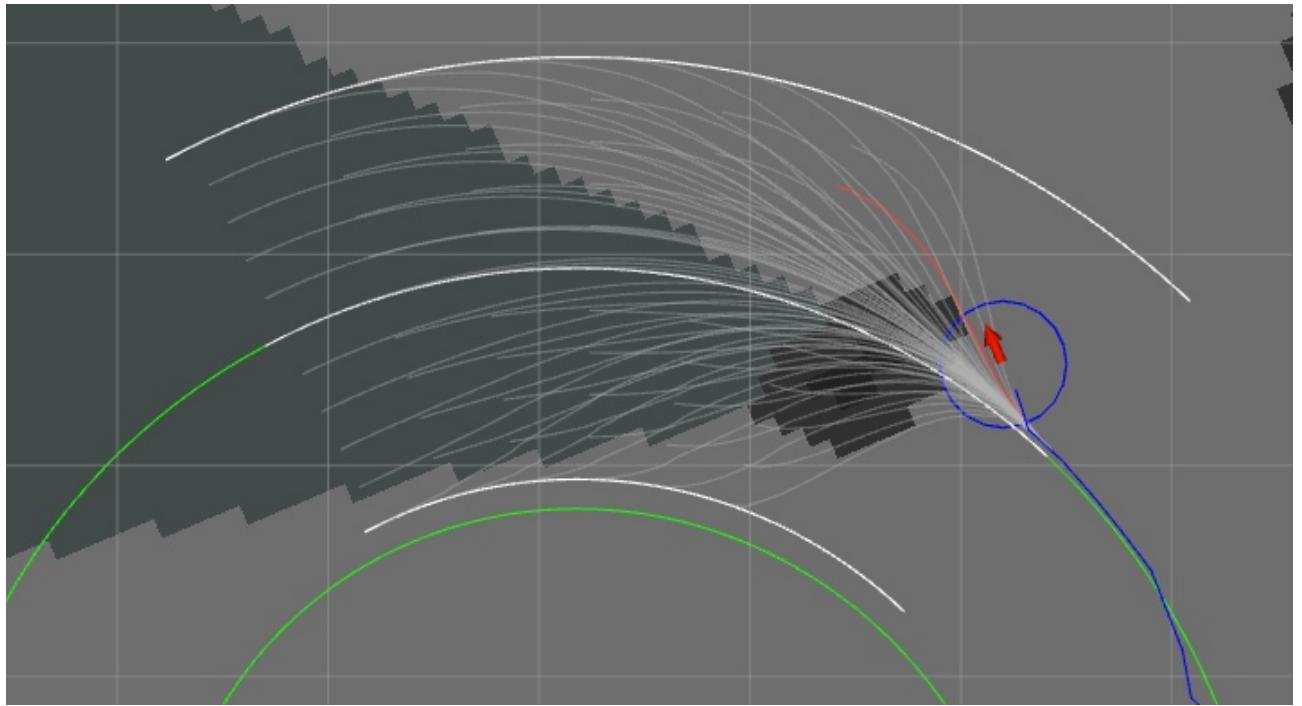


Рисунок 35 – Результаты эксперимента с движением по заданной траектории

На этом рисунке изображено движение автомобиля, обозначенного красной стрелкой, по опорной траектории (зеленая линия) в форме спирали. Толстые белые линии обозначают границу области планирования поперечных траекторий. Набор белых линий представляет траектории-кандидаты, сгенерированные алгоритмом планирования движения, а красная линия — выбранную оптимальную траекторию.

4.3 Выводы по главе

Было проведено экспериментальное исследование работы различных подсистем системы управления движением беспилотного автомобиля (системы обнаружения препятствий, системы движения по траектории, системы планирования локального движения) как по отдельности, так и совместно в составе системы управления движением. Полученные результаты свидетель-

ствуют о работоспособности разработанных алгоритмов.

ЗАКЛЮЧЕНИЕ

В данной работе была поставлена и достигнута цель, связанная с разработкой системы построения программной траектории и регулятора движения по ней беспилотного наземного транспортного средства.

В ходе работы были рассмотрены основные методы планирования движения беспилотных автомобилей и других автономных наземных транспортных средств. Проведен анализ способов методов планирования движения и выделен наиболее подходящий для использования в разработке системы управления.

На основе данных анализа литературных источников был разработан алгоритм построения программной траектории для движения к заданной цели и обездом препятствий и регулятор для движения по программной траектории с обратной связью.

Используя предложенный подход была спроектирована и реализована система управления движением беспилотного транспортного средства. Была собрана мобильная колесная платформа, оснащенная датчиками системы компьютерного зрения (стереокамерой и LiDAR) с целью экспериментальной проверки разработанных алгоритмов. Была разработана подсистема компьютерного зрения, осуществляющая обнаружение препятствий с использованием облаков точек.

Было проведено экспериментальное исследование работы системы построения программной траектории и регулятора движения по ней беспилотного транспортного средства, которое показало работоспособность разработанных решений.

Научная новизна работы заключается в следующих положениях:

- 1) был разработан алгоритм построения программной траектории для наземного беспилотного транспортного средства, осуществляющий построение траектории в форме полиномов пятого порядка, отличающейся от существующего добавлением поиском кратчайшего пути на графике состояний, что позволяет осуществлять планирование на несколько шагов вперед;

2) был разработан регулятор для движения по программной траектории с использованием обратной связи от системы SLAM.

Дальнейшее улучшение возможностей системы построения программных траекторий может быть осуществлено путем использования программы "ФРУНД" для проверки и/или формирования участков траектории с целью достижения лучшего учета динамических ограничений транспортного средства.