МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра компьютерных технологий и систем

БУШЛЯКОВА Маргарита Дмитриевна

ОПТИМИЗАЦИЯ АЛГОРИТМОВ РАСПОЗНАВАНИЯ ДОРОЖНОЙ РАЗМЕТКИ ДЛЯ РАБОТЫ В СЛОЖНЫХ ПОГОДНЫХ УСЛОВИЯХ

Дипломная работа

Научный руководитель:
кандидат педагогических наук
А. А. Францкевич

Допущена к защите
«__» _____ 2025 г.
Заведующий кафедрой компьютерных технологий и систем
доктор педагогических наук, профессор В. В. Казаченок

Минск, 2025

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

CPA — метрика для сравнения выравнивания направления движения машины с направлением дороги (Cosine Path Alignment).

HSV — цветовое пространство, основанное на трёх компонентах: оттенок, насыщенность и значение яркости (Hue, Saturation, Value).

LDE — средняя ошибка отклонения от центра полосы (Lane Departure Error).

ML — машинное обучение (Machine Learning).

MDP — марковский процесс принятия решений (Markov Decision Process).

PPO — алгоритм оптимизации стратегии в обучении с подкреплением (Proximal Policy Optimization).

Q-обучение — алгоритм обучения с подкреплением, использующий функцию ценности действия (Q-learning).

ReLU — линейная выпрямляющая функция активации (Rectified Linear Unit).

RL — обучение с подкреплением (Reinforcement Learning).

ROI — область интереса (Region Of Interest)

SAE — общество автомобильных инженеров (Society of Automotive Engineers).

UMAP — метод аппроксимации объединённых многообразий (Uniform Manifold Approximation and Projection).

Webots — среда моделирования роботов и автономных систем.

РЕФЕРАТ

Дипломная работа, 61 с., 9 рис., 22 источника.

Ключевые слова: ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ, АВТОПИЛОТИРУЕМЫЙ АВТОМОБИЛЬ, НЕЙРОННЫЕ СЕТИ, PPO, СКВОЗНАЯ АРХИТЕКТУРА, МАРКОВСКИЕ ПРОЦЕССЫ, WEBOTS, DEEPBOTS.

Объект исследования — алгоритмы компьютерного зрения, методы машинного обучения (включая глубокое обучение,обучение с подкреплением, обучение с учителем и без учителя), симуляционная среда Webots, алгоритмы распознавания дорожной разметки, архитектуры систем для автопилотирования автомобилей.

Цель работы — исследование и оптимизация алгоритмов управления движением автомобиля в сложных погодных условиях с использованием методов машинного обучения и моделирования в виртуальной среде.

Методы исследования — реализация алгоритмов компьютерного зрения на основе OpenCV, разработка систем управления движением в симуляторе Webots, проектирование сквозной архитектуры системы автопилота на основе обучения с подкреплением, обучение нейросетей с учителем, проведение симуляционных экспериментов в виртуальной среде среде.

Результаты работы — разработан и протестирован алгоритм управления движением автомобиля с применением методов анализа изображений; построена модель автопилота на основе обучения с подкреплением, способная самостоятельно обучаться вождению в сложных погодных условиях. Выявлены преимущества и ограничения различных подходов построения систем автопилота, намечен путь дальнейшего обобщения модели автопилота на всё более сложные условия.

Область применения — автономное управление транспортными средствами, системы помощи водителю, робототехника, разработка интеллектуальных систем управления в условиях ограниченной видимости, обучение и тестирование моделей в виртуальных средах без риска для людей и техники.

РЭФЕРАТ

Дыпломная работа, 61 с., 9 мал., 22 крыніцы.

Ключавыя словы: НАВУЧАННЕ З ПАДМАЦАВАННЕМ, АЎТАПІЛОТНЫ АЎТАМАБІЛЬ, НЕЙРОННЫЯ СЕТКІ, PPO, СКРОЗНАЯ АРХІТЭКТУРА, МАРКАЎСКІЯ ПРАЦЭСЫ, WEBOTS, DEEPBOTS.

Аб'ект даследавання — алгарытмы камп'ютарнага зроку, метады машыннага навучання (уключна з глыбокім навучаннем, навучаннем з падмацаваннем, навучаннем з наглядам і без нагляду), сімуляцыйнае асяроддзе Webots, алгарытмы распазнання дарожнай разметкі, архітэктуры сістэм аўтапілотнага кіравання аўтамабілямі.

Мэта працы — даследаванне і аптымізацыя алгарытмаў кіравання рухам аўтамабіля ў складаных метэаралагічных умовах з выкарыстаннем метадаў машыннага навучання і мадэлявання ў віртуальным асяроддзі.

Метады даследавання — рэалізацыя алгарытмаў камп'ютарнага зроку на базе OpenCV, распрацоўка сістэм кіравання рухам у сімулятары Webots, праектаванне скрознай архітэктуры сістэмы аўтапілота на аснове навучання з падмацаваннем, навучанне нейросетак з наглядам, правядзенне сімуляцыйных эксперыментаў у віртуальным асяроддзі.

Вынікамі з'яўляюцца — распрацаваны і пратэставаны алгарытм кіравання рухам аўтамабіля з прымяненнем метадаў аналізу выявы; пабудавана мадэль аўтапілота на аснове навучання з падмацаваннем, здольная самастойна навучацца кіраванню ў складаных метэаралагічных умовах. Выяўлены перавагі і абмежаванні розных падыходаў да пабудовы сістэм аўтапілотнага кіравання, вызначаны напрамкі далейшага абагульнення мадэлі аўтапілота на ўсё больш складаныя ўмовы.

Вобласць ужывання — аўтаномнае кіраванне транспартнымі сродкамі, сістэмы дапамогі кіроўцу, робататэхніка, распрацоўка інтэлектуальных сістэм кіравання ў умовах абмежаванай бачнасці, навучанне і тэставанне мадэляў у віртуальных асяроддзях без рызыкі для людзей і тэхнікі.

SUMMARY

Diploma thesis, 61 pages, 9 fig., 22 sources.

Keywords: REINFORCEMENT LEARNING, AUTONOMOUS VEHICLE, NEURAL NETWORKS, PPO, END-TO-END ARCHITECTURE, MARKOV PROCESSES, WEBOTS, DEEPBOTS.

The object of research is computer vision algorithms, machine learning methods (including deep learning, reinforcement learning, supervised and unsupervised learning), Webots simulation environment, lane detection algorithms, architectures of autonomous driving systems.

The purpose of the work is to investigate and optimize vehicle control algorithms in challenging weather conditions using machine learning methods and modeling in a virtual environment.

The research methods are implementation of computer vision algorithms based on OpenCV, development of vehicle control systems in the Webots simulator, design of an end-to-end autonomous driving system architecture based on reinforcement learning, supervised neural network training, and conducting simulation experiments in a virtual environment.

The results are an algorithm for lane-keeping with obstacle avoidance was developed and tested in a simulator using image analysis methods and a reinforcement learning-based autonomous driving model capable of self-learning to drive in challenging weather conditions was built. Advantages and limitations of different approaches to autonomous driving system design were identified, and directions for further generalization of the autonomous driving model to increasingly complex conditions were outlined.

The scope is autonomous vehicle control, driver assistance systems, robotics, development of intelligent control systems under limited visibility conditions, training and testing of models in virtual environments without risk to humans or equipment.

ВВЕДЕНИЕ

По данным всемирной организации здравоохранения [10], ежегодно в Беларуси погибает порядка 600–700 человек в дорожно-транспортных происшествиях (приблизительно 7.4 на 100 000 человек), а во всем мире более миллиона человек, и до 90% ДТП происходит по вине человека (превышение скорости, усталость, алкоголь, отвлечение). Автопилотируемые автомобили имеют большой потенциал в улучшении безопасности на дорогах. Системы автономного вождения могут реагировать быстрее человека, они видят во все стороны благодаря камерам и лидарам, и предсказывают поведение других участников, они поддерживают безопасную дистанцию и скорость и могут быстро обмениваться данными друг с другом.

Исследования в области автоматизации автомобилей ведутся с 1984, начиная с навигационной лаборатории университета Карнеги — Меллона. Их проект представлял собой фургон с суперкомпьютером и учёными на борту. Фургон двигался по пустым дорогам с максимальной скоростью в 32 километра, и использовал для навигации датчики расстояния и видеокамеры, а также однослойную нейронную сеть, которая определяла углы поворота, используя данные с датчиков. Технология, используемая учёными лаборатории, заложила основы для дальнейшего развития автопилотируемых автомобилей. Вернёмся в наши дни, сейчас термин «автопилотируемые автомобили» и другие подобные термины употребляются очень широко, и в связи с этим, прежде чем перейти к постановке задачи, важно определиться, как будет пониматься этот термин в работе. Так, общество автомобильных инженеров (SAE) в 2014 году разработало классификацию автоматизации автомобилей [11]. Существует разделение на 6 уровней:

0. Никакой автоматизации;

1. Водитель управляет автомобилем, но ему в помощь предоставлены некоторые системы (например, автоматическая парковка);

2. Частичная автоматизация (ускорение, торможение, управление рулём), но при этом водитель должен всегда «присматривать». Автомобили BMW, Ford, Tesla оснащены этой системой;

3. Система сама реагирует на ситуации, требующие немедленных действий, таких как экстренное торможение, но водитель должен быть готов в любой момент взять на себя управление автомобилем, получив уведомление. Первым производителем, продавшим автомобиль этого уровня, стал автоконцерн Honda в 2021 году. В 2023 году такая система появилась в автомобилях Mercedes-Benz;

4. В определённых условиях автопилот может полностью выполнять управление, водитель может покинуть место пилота. Компания Waymo в 2020 году первой в мире предложила услуги беспилотных такси этого уровня;

5. В любых условиях помощь водителя не требуется. По состоянию на начало 2025 года ни одна система не достигла этого уровня.

Таким образом задача создания системы автопилотирования, очень обширная. Она включает в себя множество фундаментальных задач, таких как, обнаружение проезжей части, дорожной разметки, перекрёстков, пешеходных переходов, других транспортных средств, пешеходов, животных, предметов на дороге, дорожных знаков и светофоров. А также, определение угла поворота руля, положения педали газа и т. д. на основе данных, поступающих с датчиков. Кроме того, выполнение обгонов, смена полосы движения, развороты, выезд на дорогу и съезд с неё, пересечение перекрёстков с круговым движением, проезд по туннелям, мостам. Также важно реагировать на нарушения правил дорожного движения другими водителями, неожиданные изменения в окружающей обстановке, перемещаться в условиях пробок. И многое другое. В связи с этим, в этой работе мы сконцентрируемся лишь на нескольких из них, а именно на задаче распознавания проезжей части, задаче удержания полосы движения, путём управления рулём и положением педали газа, на основе данных поступающих с датчиков. Немного коснёмся задачи детектирования и объезда препятствий.

Целью работы является изучение алгоритмов компьютерного зрения, глубокого обучения, обучения с подкреплением для решения задач автопилотирования автомобиля.

В главе 1 рассмотрено как можно реализовать простую систему удержания полосы движения в ограниченных условиях с использованием лишь классических алгоритмов компьютерного зрения для выделения разметки на изображении и удержании полосы движения исходя из этих данных и показно как можно расширить такую систему для возможности объезда препятствий за счёт добавления новых датчиков используя иерархичный подход проектирования системы автопилота

В главе 2 рассмотрены основные виды машинного обучения и проанализирована возможность их применения для решения задачи в более широких условиях в том числе при плохих погодных условиях, в потоке движения автомобилей, в сложной местности, а также рассмотрены варианты архитектуры системы автопилота у лидеров рынка беспилотных автомобилей.

Глава 3 посвящена разработке системы автопилота, способной самостоятельно принимать решения в ограниченных условиях, а также такой, которую можно удобно и быстро расширять на всё более сложные условия, включая сложные погодные условия, неожиданные ситуации на дороге, отсутствие дорожной разметки и прочее. Система основана на обучении с подкреплением и сквозной архитектуре, что позволяет ей потенциально научиться управлять автомобилем лучше водителя человека, и не требует для своего обучения заранее собранных данных.

ГЛАВА 1
СИСТЕМА УДЕРЖАНИЯ ПОЛОСЫ ДВИЖЕНИЯ, ОСНОВАННАЯ НА АЛГОРИТМАХ АНАЛИЗА ИЗОБРАЖЕНИЙ

1.1 Постановка задачи

Рассмотрим задачу управления направлением движения для удержания полосы дорожного движения. И введём для начала некоторые ограничения:

1. На дороге отсутствуют перекрёстки, другие автомобили, дорожные знаки, препятствия.
2. Автомобиль использует единственную камеру, установленную на капоте.
3. Дорога не меняет уклон, а только лишь направление.
4. На дорогу нанесена только дорожная разметка полос движения.

Рассмотрим возможные подходы к решению поставленной задачи.

Первый подход заключается в том, чтоб анализировать непосредственно изображение, получаемое с камеры методами компьютерного зрения и, используя симуляцию в виртуальной среде, путём экспериментов разработать алгоритм, который будет определять необходимое направления движения, основываясь на распознавании полос дорожной разметки.

Второй подход заключается в том, чтоб при минимальной обработке изображения передавать его в нейронную сеть, которая, используя большое количество данных, обучится выбирать направления движения.

Эта глава посвящена первому подходу, а следующие две рассмотрят второй.

Итак, перед нами стоит задача распознавания полос дорожной разметки по изображению. А угол наклона этих полос подскажет нам направление движения дороги.

1.2 Алгоритмы компьютерного зрения для выделения объектов на изображении

1.2.1 Фильтрация изображений с помощью масок

Нам необходимо каким-либо образом выделить линии дорожной разметки, которые очерчивают полосу, эту задачу можно разделить на несколько этапов. Во-первых, нам нужно отделить линии, которые обычно являются белыми или жёлтыми от окружающего их асфальта, во-вторых, требуется заполнить небольшие пробелы на линиях вызванные облезанием

краски и прочим. Третий этап — это выделение из близкорасположенных белых точек линий определённой длины. И четвёртый этап — это необходимость аппроксимировать отдельно левые и правые линии в некие две направляющие для полосы движения.

В этом пункте рассмотрим первый этап, а именно выделение линий по цвету. Понятно, что линии будут самыми светлыми на дороге точками, поэтому для этой задачи мы хотим выделить все пиксели, светлота которых превышает некоторый установленный порог светлоты, и в этом помогает алгоритм применения масок. Маска позволяет выделить на изображение нужное и отбросить остальное (зачернить). Она представляет собой бинарное изображение, где 0 — это отсутствие интереса (чёрный цвет), а 1 — область интереса (белый цвет). Рассмотрим, как она работает.

Пусть I — это исходное изображение, представленное как матрица пикселей, где каждый элемент $I(x, y)$ представляет собой значение интенсивности пикселя на позиции (x, y). Маска M — это бинарная матрица того же размера, что и изображение I, где каждый элемент $M(x, y)$ равен 1 для области интереса и 0 для неинтересной области. Для применения маски к изображению можно использовать формулу (1.1).

$$I_{masked}(x, y) = I(x, y) \cdot M(x, y), \tag{1.1}$$

где $\mathrm{I_{masked}(x, y)}$ — это результат применения маски.

Если $\mathrm{M(x, y) = 1}$, то пиксель из исходного изображения сохраняется, иначе пиксель скрывается, в нашем случае $\mathrm{M(x, y) = 1}$ у тех пикселей, светлота которых выше определённого заранее заданного порога.

Удобно работать с изображением в формате HSV, этот формат обладает некоторой инвариантностью к изменениям освещения, также в HSV легко настраивать пороговые значения для выделения нужных цветов. Например, для выделения белого цвета достаточно установить пороги для компоненты значения в диапазоне, соответствующем белому цвету

1.2.2. Применение эрозии и дилатации для удаления шума, выделения объектов, заполнения пробелов и разделения объектов

Перейдём ко второму этапу, а именно, к устранению некоторых разрывов в линиях, заполнению пустых мест на нарисованной краской линии разметки для того, чтоб можно было позже по точкам выделить направляющую линию достаточной длины, кроме того, нам может потребоваться устранить слишком маленькие белые участки после применения маски и убрать возможные лишние детали на дороге. Помочь с

этой задачей может применение эрозии и дилатации. Рассмотрим суть этих операций.

Эрозия применяется для уменьшения размера объектов на изображении. Операция применяется к каждому пикселю на изображении и его окружению с помощью некоторого структурирующего элемента (обычно квадратного или круглого). Если хотя бы один пиксель в окне структурирующего элемента является чёрным (нулевым), то центральный пиксель на изображении также становится чёрным. Иначе он остаётся белым. Эрозия может использоваться для удаления шума и мелких деталей на изображении, а также для разделения объектов, связанных узкими областями.

Эрозию можно определить так: пусть I(x, y) — это изображение, а S — структурирующий элемент. Операция эрозии выполняется по формуле (1.2).

$$(I \circ S)(x, y) = min\, I(x + a, y + b), \text{на } (a, b) \in S, \qquad (1.2)$$

где $I(x, y)$ — значение пикселя изображения в точке (x, y),
S — структурирующий элемент, который накладывается на изображение, (a, b) — смещение в пределах структурирующего элемента S.

Этот процесс уменьшает объект, так как центральный пиксель изображения будет чёрным (нулевым), если хотя бы один пиксель в окне структурирующего элемента чёрный.

Дилатация применяется для увеличения размера объектов на изображении. Операция применяется к каждому пикселю на изображении и его окружению с использованием структурирующего элемента аналогично эрозии. Она выполняет ровно обратное эрозии. Дилатация используется для заполнения пробелов между объектами, объединения близко расположенных объектов и заполнения небольших областей в объектах.

Дилатацию можно определить по формуле (1.3).

$$(I \oplus S)(x, y) = max\, I(x + a, y + b), \text{на } (a, b) \in S, \qquad (1.3)$$

где $I(x, y)$ — значение пикселя изображения в точке (x, y),
S — структурирующий элемент, который накладывается на изображение, (a, b) — смещение в пределах структурирующего элемента.

Этот процесс увеличивает объект, так как центральный пиксель изображения будет белым (максимум), если хотя бы один пиксель в окне структурирующего элемента белый.

Для более сложной обработки изображений удобно комбинировать операции эрозии и дилатации. Например, последовательное применение

эрозии и дилатации помогает удалить шум и затем соединить или заполнить пробелы между объектами. Такое сочетание операций может быть полезным для различных задач предварительной обработки изображений перед более сложными операциями, такими как сегментация, выделение контуров или распознавание объектов. В нашей задаче мы сначала заполняем пробелы в раскраске линий дорожной разметки, а после уменьшаем их (сужаем), чтоб проще было далее выделить направляющие линии.

1.2.3. Преобразование Хафа

Как было сказано ранее, третьим этапом требуется выделить из белых пикселей, которые находятся рядом, линии, такие, чтоб их длина была больше заданного порога (это нужно для того, чтоб мы выделили именно «вертикальные» направляющие линии дороги, а не просто линии проведённые поперёк нарисованной полоски разметки или как угодно иначе), нас интересуют достаточно длинные линии, или же мы можем провести все, а затем среди них искать самые длинные слева и справа. Для обнаружения прямых линий, а нас интересуют именно прямые линии (даже если дорога изгибается, все равно часть направляющей будет прямой), поскольку дальше нам будет так проще определить их наклон и принять решение о повороте, мы можем использовать алгоритм Хафа.

Преобразование Хафа — это классический алгоритм обработки изображений, который используется для обнаружения формы или шаблона, представленных на изображении, разработанный ещё в 60-ых [4]. Рассмотрим основные этапы алгоритма:

Переход координат в пространство Хафа. Каждая точка изображения с координатами (x, y) может быть представлена в пространстве Хафа с помощью параметров ρ и θ, где ρ — это расстояние от начала координат до прямой, θ — это угол между вертикальной осью и прямой, которая проходит через точку (x, y). Преобразование для каждой точки изображения (x, y) в параметры (ρ, θ) задается формулой (1.4):

$$\rho = x \cdot cos\,(\theta) + y \cdot sin\,(\theta) \tag{1.4}$$

Далее необходимо построить аккумуляторное пространство, которое представляет собой двумерный массив или матрицу, где каждый элемент соответствует возможной комбинации параметров ρ и θ. Каждый пиксель изображения, который может принадлежать линии, "голосует" за параметры (ρ, θ) в аккумуляторном пространстве. Чем больше пикселей проголосовало за определённые параметры, тем более вероятно, что эти параметры

соответствуют настоящей линии. Аккумуляторное пространство можно записать как $A(\rho, \theta)$, где каждый элемент $A(\rho, \theta)$ содержит количество голосов за параметры ρ и θ.

После того как все пиксели проголосуют за параметры, пиковые точки в аккумуляторном пространстве будут соответствовать тем параметрам, которые наиболее часто встречаются в изображении. Пиковые точки в аккумуляторном пространстве указывают на присутствие прямых линий на изображении. Если $A(\rho, \theta)$ является значением в аккумуляторном пространстве для параметров (ρ, θ), то линия будет обнаружена, если $A(\rho, \theta)$ превышает некоторый порог T (1.5):

$$A(\rho, \theta) > T, \tag{1.5}$$

где T — это пороговое значение для принятия параметров как наличия линии.

Пиковые точки в аккумуляторном пространстве соответствуют параметрам прямых линий в пространстве Хафа. Каждая точка в аккумуляторном пространстве имеет параметры (ρ, θ), которые можно интерпретировать как параметры прямой, найденной на изображении.

Таким образом, параметры каждой обнаруженной прямой можно записать как (1.6):

$$\rho = x \cdot cos\,(\theta i) + y \cdot sin\,(\theta i), \tag{1.6}$$

где i — индекс каждой обнаруженной прямой.

Так, выделив прямые линии с помощью алгоритма, нам остаётся только отобрать линии, которые могут соответствовать направляющим и определить угол из наклона, для обнаружения направления дороги и принятии решения об управлении машиной.

1.3 Подготовка среды виртуального моделирования

Прежде чем перейти к разработке алгоритма, логичным шагом видится подготовить среду для его тестирования по ходу разработки на движущемся в симуляции автомобиле. Существует множество виртуальных сред, некоторые из них фотореалистичные, другие нет, но поскольку перед нами сейчас не стоит задача распознавания сложных объектов, то достаточным видится использование нефотореалистичного симулятора в целях экономии GPU. Однако это не мешает нам, взяв для примера видео с видеорегистратора

автомобиля и применив к нему алгоритм, оценить, насколько правильно распознаются линии дорожной разметки на фотореалистичном видео.

Итак, после проведённого исследования доступных средств моделирования, таких как Gazebo, Webots, фотореалистичная Carla, был выбран бесплатный симулятор роботов Webots с открытым исходным кодом, используемый в промышленности, образовании и исследованиях.

Webots позволяет создавать и тестировать программное обеспечение для различных типов роботов в виртуальной среде. Симулятор предоставляет интуитивно понятный интерфейс и богатый набор инструментов для создания симуляций, а также поддерживает различные языки программирования, такие как C++, Python, Java и MATLAB. Проекты в Webots представляют собой миры, в которые в качестве узлов можно добавлять всевозможные объекты, можно добавить различные модели роботов, окружающей среды, такие как стены, здания, объекты интерьера и экстерьера и прочее. Каждый робот в проекте может иметь свой собственный контроллер, который управляет его поведением и взаимодействием с окружающей средой.

Webots предлагает несколько готовых миров, посвящённых дорожному движению. Но было решено построить свой мир, и интегрировать на неё карту окрестностей главного корпуса Белорусского государственного университета. OpenStreetMap является онлайн ресурсом с картой мира, и может свободно использоваться под открытой лицензией. Сайт сервиса позволяет выбрать область карты и экспортировать её в формате osm. После этого карту можно доработать и отредактировать в редакторе JOSM. Запустив в командной строке, содержащейся в папке ресурсов Webots скрипт (где “myMap.osm” — это экспортированная карта, “myMap.wbt” — создаваемый мир), можно импортировать выбранную область в мир webots:

importer.py --input=myMap.osm --output=myMap.wbt

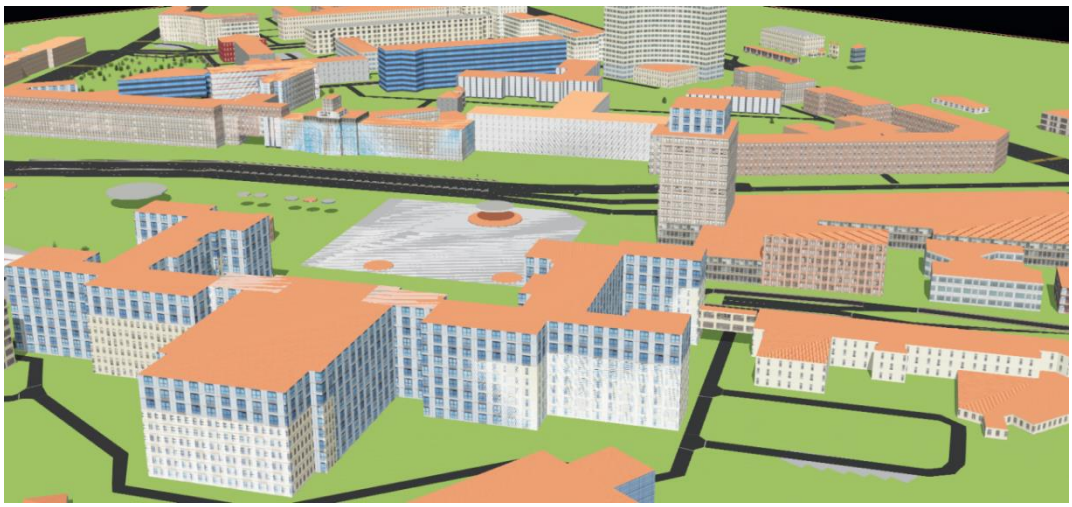

Рисунок 1.1 — Карта, построенная в Webots

На рисунке 1.1 мы видим построенный мир в Webots. Перейдём к выбору и подготовке автомобиля, настройке его контроллера. В качестве автомобиля была выбрана модель Toyota Prius, для которой была проведена модификация (рисунок 1.2), а именно установка в передний слот для датчиков трёх датчиков измерения расстояния справа, слева и по центру автомобиля соответственно и камеры. Это было сделано с расчётом на то, что датчики расстояния позволят обнаруживать препятствия на дороге и объехать их за счёт перестройки в соседнюю полосу, а камера будет отвечать за удержание полосы. Таким образом мы сможем в дальнейшем расширить нашу задачу, добавив объезд препятствий.

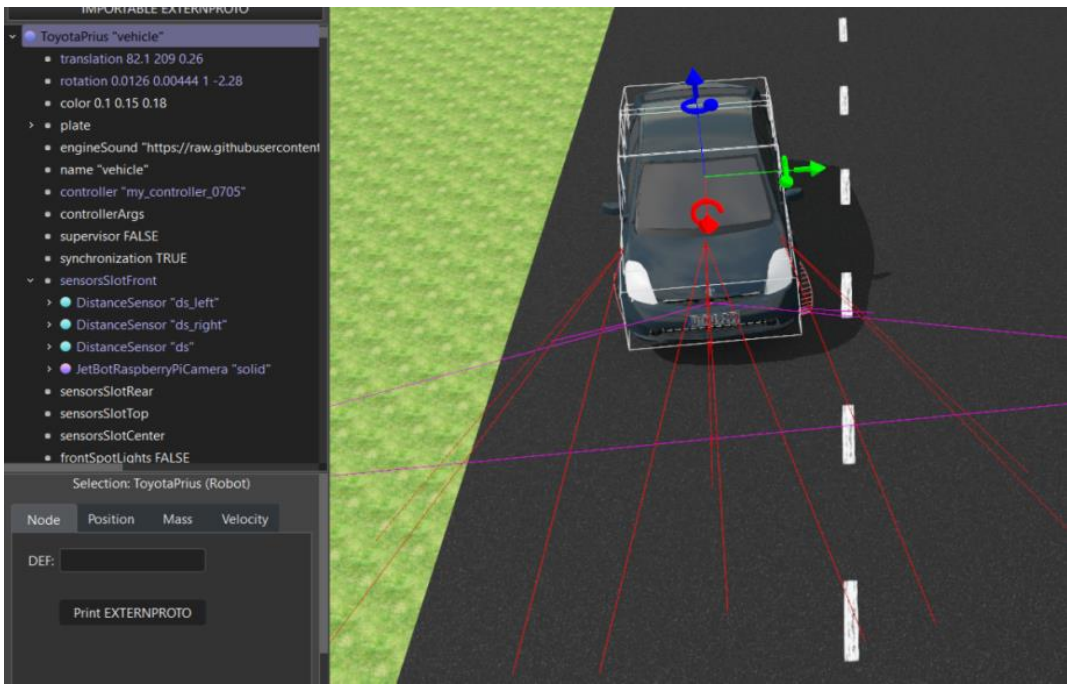

Рисунок 1.2 — Модификация автомобиля в Webots

На рисунке 1.2 можно увидеть лучи сенсоров расстояния и область захвата камеры. Управление автомобилем осуществляется с помощью контроллера на языке Python. Данные с датчиков расстояния и камеры анализируются и принимается решение об угле поворота руля автомобиля.

1.4 Разработка алгоритма управления движением автомобиля

Мы рассмотрели алгоритмы маскирования, эрозии, дилатации и преобразования Хафа для выделения прямых линий, подготовили среду. Перейдём к разработке алгоритма управления направлением движения автомобиля, основываясь на данных с камеры, установленной на капоте этого автомобиля, в условиях, которые были описаны в пункте 1.1 о постановке задачи.

Шаг 1. Начнём с захвата и предобработки кадра.

Кадр преобразуется в цветовое пространство HSV, определяется диапазон белого цвета для пороговой обработки (если есть жёлтые полосы дорожной разметки, то можно дополнительно использовать и маску жёлтого). Проходит создание маски белого (рисунок 1.3), для обнаружения белых полос разметки. В результате этого шага мы получаем черно-белое изображение, на котором присутствуют интересующие нас линии, определяющие полосу дорожного движения, по которой нужно продолжать двигаться автомобилю.

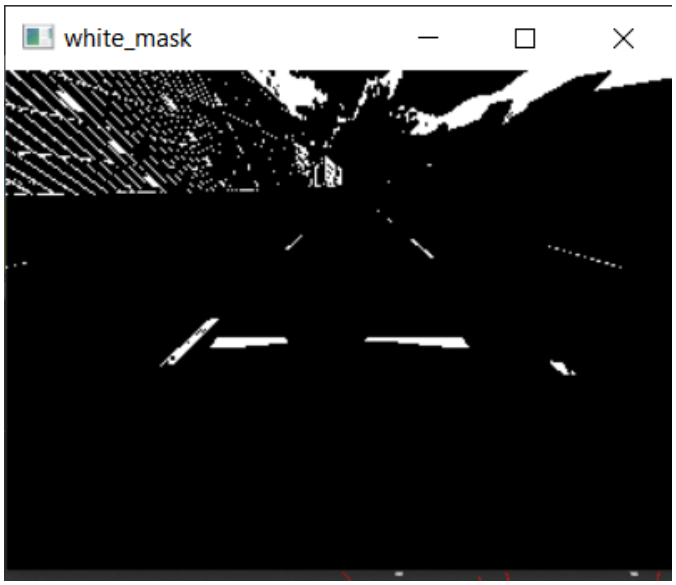

Рисунок 1.3 — Применение белой маски

Однако на этом этапе нужные линии имеют полости, могут быть недостаточно ясными, кроме того, на изображении присутствуют и другие белые объекты помимо нужных нам.

Шаг 2. Применение дилатации и эрозии.

На этом этапе мы делаем линии более чёткими (это достигается последовательным применением дилатации и эрозии к изображению) и готовим их для последующей обработки (рисунок 1.4). Применяем преобразование Хафа для нахождения линий.

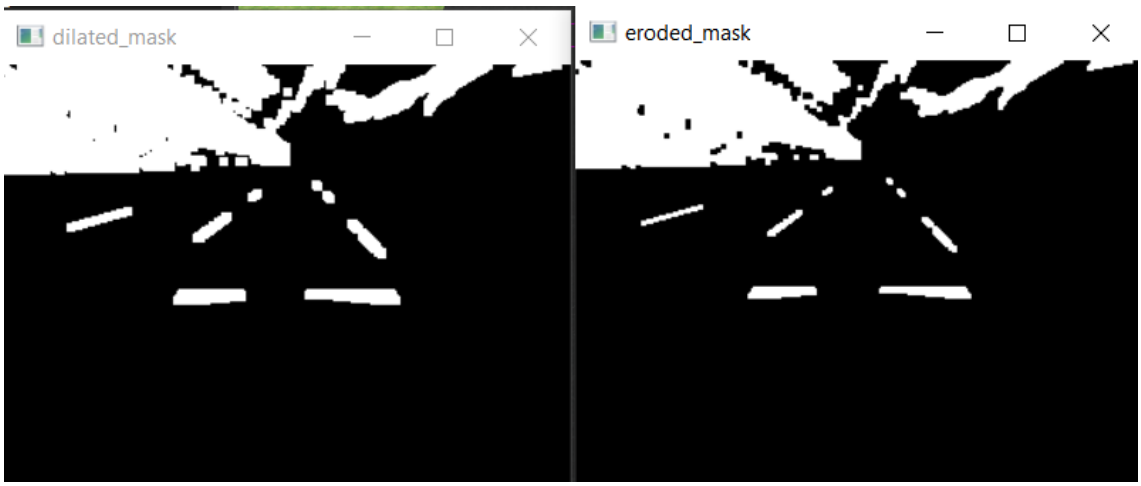

Рисунок 1.4 — Применение дилатации, а после эрозии к изображению

Код обработки изображения на шагах 1,2,4 представлен ниже:

```
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

# Определение белых областей (полос дорожной разметки) в изображении
lower_white = np.array([0, 0, 200])
upper_white = np.array([200, 50, 255])
white_mask = cv2.inRange(hsv, lower_white, upper_white)

# Применяем операцию дилатации к бинарному изображению
dilated_mask = cv2.dilate(white_mask, np.ones((5, 5), np.uint8))

# Применяем операцию эрозии к полученному изображению
eroded_mask = cv2.erode(dilated_mask, np.ones((3, 3), np.uint8))

# Применяем преобразование Хафа для обнаружения линий
lines = cv2.HoughLinesP(eroded_mask, 1, np.pi/180, 30, minLineLength=2, maxLineGap=10)
```

Шаг 3. Определение параметров области интереса.

Зададим область, в которой должны находиться найденные линии Хафа, чтоб восприниматься как линии нужной нам полосы движения автомобиля. Посмотрим на изображение. Очевидно, что в верхней части изображения дорога отсутствует, мы там видим окружающие здания, небо и прочее, что нас не интересует в рамках поставленной задачи. Также мы видим разметку соседних полос движения, что может нас отвлекать и сбивать, поэтому необходимо немного обрезать изображение и по бокам. Снизу же виден капот авто, на котором тоже обнаруживаются линии, но совсем не те, что нужны. Таким образом, мы выделяем область интереса, основываясь на том, как расположена камера. В данном случае это будет трапеция, в которой предположительно должны находиться нужные полосы разметки. Также, на этом этапе в трапеции строиться линия через центр верхнего и нижнего основания, которая в дальнейшем будет использоваться для расчёта угла наклона обнаруженных линий. Будем называть эту линию центральной. Эта линия соответствует текущему направлению движения автомобиля. А наша задача выяснить наклон направляющих линий относительно центральной.

Рисунок 1.5 — Область интереса (ROI), центральная линия, линии Хафа, две направляющие линии текущей полосы

На рисунке 1.5 мы видим трапецию области интереса, центральную линию, линии Хафа, которые мы нашли, и те из них, что удовлетворяют области интереса и являются таким образом направляющими линиями и

показывают в каком направлении поворачивает дорога (находятся на шаге 4). И на рисунке мы видим, что автомобилю следует повернуть налево.

Шаг 4. Нахождение правой и левой полос дорожной разметки.

Применим преобразование Хафа для обнаружения линий на изображении границ. Отбираем из линий Хафа те, у которых хотя бы одна из точек лежит в области интереса. Код, проверяющий принадлежность отрезка, обнаруженного преобразованием Хафа, на принадлежность ROI представлен ниже:

```
def is_line_in_roi(line, roi_vertices):
    # Распаковываем координаты вершин ROI
    roi_pts = np.array(roi_vertices[0], dtype=np.int32)
    # Получаем координаты концов отрезка
    x1, y1, x2, y2 = line[0]

# Проверяем, что хотя бы один конец отрезка или его середина находятся внутри ROI
    return (cv2.pointPolygonTest(roi_pts, (float(x1), float(y1)), measureDist=False) >= 0 or
            cv2.pointPolygonTest(roi_pts, (float(x2), float(y2)), measureDist=False) >= 0 or
            cv2.pointPolygonTest(roi_pts, ((float(x1) + float(x2)) // 2, (float(y1) + float(y2)) // 2), measureDist=False) >= 0)
```

Функция cv2.pointPolygonTest, проверяет принадлежит ли точка области.

Такой код позволяет достаточно точно выбрать из всех линий только нужные нам.

Шаг 5. Выбор правой и левой линий полосы.

Среди оставшихся линий чей центр лежит по правую сторону от центральной находим линию, угол между которой и центральной наименьший. Это нужно для того, чтоб в случае, если в область интереса попадут линии соседней полосы, предпочтение отдавалось линиям текущей полосы. Код алгоритма, по которому определяется угол между линиями (центральной и правой) приведён ниже.

```
def angle_between_lines(line1, line2):
    if line1 is None or line2 is None:
        return 90  # Возвращаем угол 90, если одна из линий не обнаружена
    x1, y1, x2, y2 = line1[0]
```

```
x3, y3, x4, y4 = line2[0]
dx1 = x2 - x1
dy1 = y2 - y1
dx2 = x4 - x3
dy2 = y4 - y3
dot_product = dx1 * dx2 + dy1 * dy2
magnitude1 = np.sqrt(dx1**2 + dy1**2)
magnitude2 = np.sqrt(dx2**2 + dy2**2)
if magnitude1 == 0 or magnitude2 == 0:
    return 90  # Возвращаем угол 90, если одна из линий имеет нулевую
длину
cos_theta = dot_product / (magnitude1 * magnitude2)
angle = np.degrees(np.arccos(np.clip(cos_theta, -1.0, 1.0)))
if angle > 90:
    angle = 180 - angle
return angle
```

Аналогично для линий слева. Таким образом, получаем правую и левую апроксимированные полосы. Чтоб понять является линия правой или левой смотрим на расположение её центра (где x1, x2 - координаты текущей линии(отрезка) по x; center_line_x1, center_line_x2 координаты центральной линии(отрезка) по x):

```
# Рассчитываем отклонение линии от центральной линии
    deviation = ((x1 + x2) // 2) - ((center_line_x1 + center_line_x2) // 2)
```

Шаг 6. Принятие решения о повороте машины на основании данных с камеры.

Анализируются острые углы между правой/левой линиями и центральной. Как мы видим на рисунке 1.5, острый угол между правой линией и центральной меньше, чем между левой и центральной на некоторое количество градусов и исходя из этого принято решение о повороте налево, для того, чтоб выровняться в полосе. Система принимает решение о направлении движения автомобиля (поворот вправо, влево или ехать прямо). В зависимости от конкретной разности углов принимается решение о плавном или более резком повороте. Это можно отрегулировать углом поворота руля.

Таким образом, мы разработали алгоритм управления направлением движения, основанной на анализе изображений с использованием библиотеки OpenCV. В следующем пункте мы попробуем пойти дальше и

научить автомобиль обнаруживать препятствия на дороге с помощью лидаров, объезжать их перестраиваясь в соседнюю полосу и продолжать движение по ней.

1.5 Интеграция обработки датчиков расстояния в получившуюся систему, для возможности объезда препятствий.

Для обнаружения и объезда препятствий на дороге, мы добавили в автомобиль датчики расстояния. В данной главе мы используем иерархичную архитектуру системы: вручную прописываем взаимодействие между модулем распознавания дорожной разметки и модулем обнаружения препятствий, в последующих главах, когда мы будем использовать нейронные сети, взаимодействие между модулями будет происходить напрямую (то есть выходы одного модуля передаются напрямую на вход следующему). Подробнее об этом в главе 2. Таким образом при обнаружении препятствий на дороге, автомобиль может своевременно на них реагировать и перестраиваться в другую полосу, а после за счёт камеры, выравниваться в этой полосе и продолжать движение. Поэтому необходимо отдать данным с датчиков расстояния больший приоритет по сравнению с изображением с камеры. Это означает, что сначала мы анализируем информацию с датчиков и в случае отсутствия препятствий переходим к алгоритму удержания полосы используя камеру. Кроме того, во время перестройки в другую полосу, может возникнуть ситуация, когда линии временно не обнаружены, для этого хранится предыдущее решение о направлении движения от камеры. Ниже приведён код принятие решения о повороте на основе анализа изображения.

```
# Определяем направление движения на основе углов между линиями
    if np.isfinite(min_left_angle) and np.isfinite(min_right_angle):
        if min_left_angle < min_right_angle - 10:
            direction = "right"#если смотреть на машину сзади
        elif min_right_angle < min_left_angle - 10:
            direction = "left"#если смотреть на машину сзади
        else:
            direction = "straight"
    else:
        direction = prev_direction  # Используем предыдущее значение направления, если углы не конечны
        # Обновляем значение предыдущего направления
        prev_direction = direction
```

Общий алгоритм управления рулём на основе камеры и датчиков расстояния приведён ниже. В случае, если расстояние до препятствия меньше 5.5 метров выполняем объезд.

```
    if distance < 5.5 or left_distance < 5.5 or right_distance < 5.5 :
        print("на центральном датчике:", distance)
        print("на правом датчике:", right_distance)
        print("на левом датчике:", left_distance)
        if left_distance < 5.5:
            print("Обнаружено препятствие слева, выполняем поворот направо")
            driver.setSteeringAngle(0.8)
        else:
            print("Обнаружено препятствие справа, выполняем поворот налево")
            driver.setSteeringAngle(-0.8)
      else:
        # Если нет препятствий, управляем движением на основе обнаруженных камерой линий
        if direction == "left":
            print("Поворот на лево")
            if abs(min_right_angle - min_left_angle) > 25:
                driver.setSteeringAngle(-1)  # Резкий поворот
            elif abs(min_right_angle - min_left_angle) > 20:
                driver.setSteeringAngle(-0.6)
            elif abs(min_right_angle - min_left_angle) > 10:
                driver.setSteeringAngle(-0.2)  # более плавный
        elif direction == "right":
            print("Поворот на право")
            if abs(min_right_angle - min_left_angle) > 25:
                driver.setSteeringAngle(1)  # Резкий поворот
            elif abs(min_right_angle - min_left_angle) > 20:
                driver.setSteeringAngle(0.6)
            elif abs(min_right_angle - min_left_angle) > 10:
                driver.setSteeringAngle(0.2)
        else:
            driver.setSteeringAngle(0.0)  # Ехать прямо
```

Таким образом, наш автомобиль способен двигаться по дороге без перекрёстков и прочего, удерживать свою полосу и объезжать препятствия.

Понятно, что этот алгоритм не универсален и действует только в определённых условиях, может применяться только на дорогах с чёткой конкретной разметкой, отсутствием другого транспорта и прочее. Ему составит затруднение выпавший на дорогу снег и другие плохие погодные условия. Но он способен провести машину по дороге (хоть из-за того, что он позволяет лишь семь дискретных значений для поворота руля, движение может быть немного дёрганным). Алгоритм хорош своей простотой и понятностью. В следующем пункте мы протестируем его в симуляторе.

1.6 Проведение экспериментов в среде моделирования Webots, их анализ

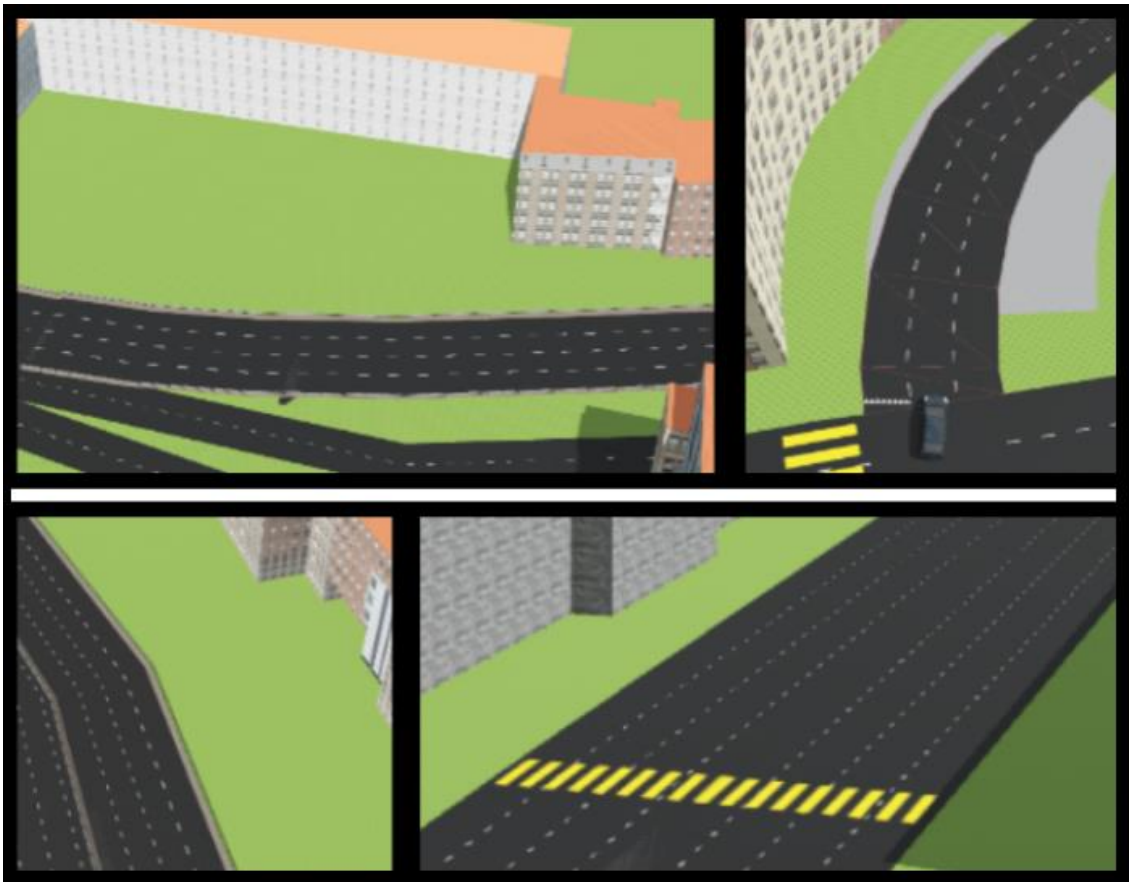

Рисунок 1.6 — Участки дороги на которых проводилось тестирование

Как было упомянуто раннее в пункте 1.3, мы построили карту окрестностей Белорусского Государственного Университета для проведения экспериментов и определению необходимых углов поворота руля при различных поступающих данных. На рисунке 1.6 вверху слева мы видим главный корпус БГУ со слегка изогнутой дорогой, а справа соседнюю дорогу с более сильным, но плавным изгибом, на рисунке 1.6 внизу — дорога с

резким поворотом слева, а справа дополнительные линии на разметке, которые могли бы помешать движению.

Для наглядности в процессе движения на экран выводится картинка с камеры автомобиля на которой нарисована область интереса (область, внутри которой лежит полоса, по которой мы движемся) и обнаруженные линии дорожной разметки, а также центральная ось, которая показывает текущее направление движения автомобиля. Кроме того, на изображение выводится направление поворота, который машина собирается осуществить. Пример такого вывода можно увидеть ниже на рисунке 1.7.

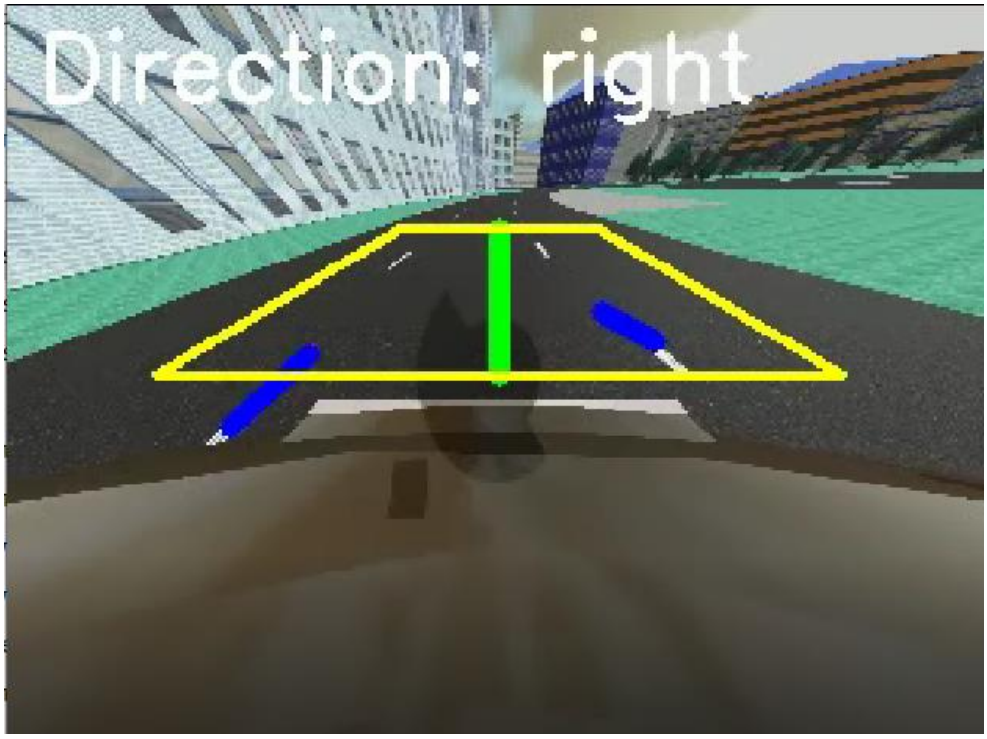

Рисунок 1.7 — Картинка с камеры автомобиля, обработанная системой

Так, в ходе ряда экспериментов были установлены достаточно хорошие углы поворота рулевого колеса автомобиля, в зависимости от наличия препятствий, разницей между углами левой/правой линии и центральной линией ROI.

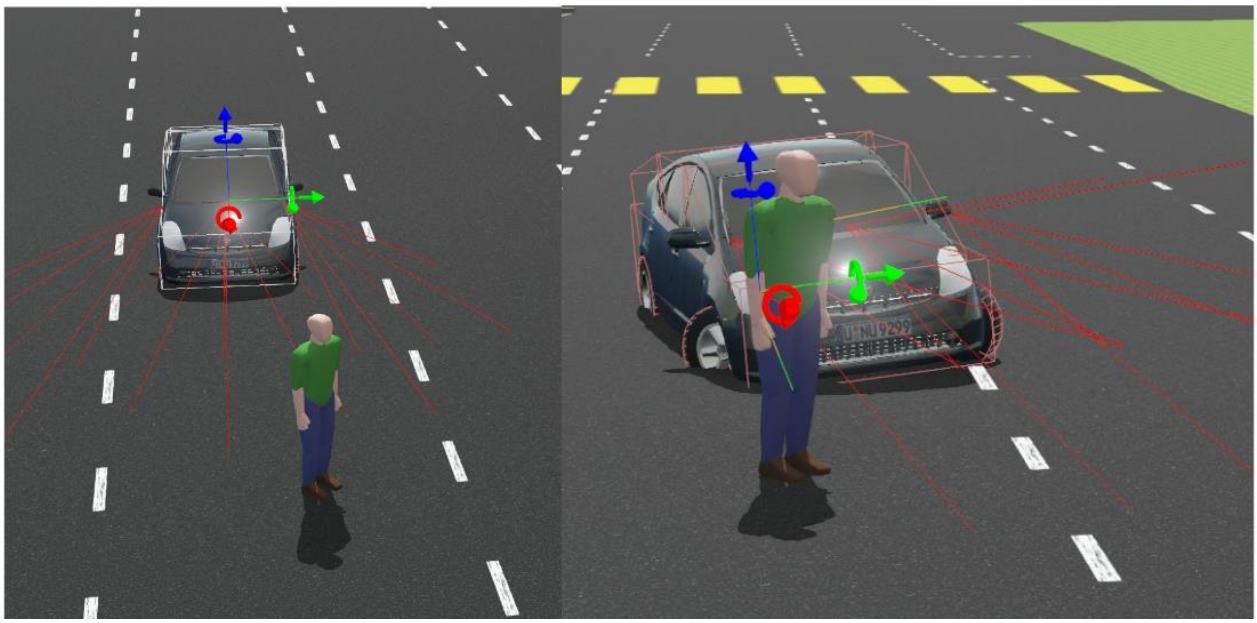

Рисунок 1.8 — Человек на дороге, выполняется объезд

Теперь посмотрим на эксперимент по обнаружению человека на дороге и его объезду с перестройкой на соседнюю полосу (рисунок 1.8). В ходе проведения экспериментов алгоритм хорошо себя показал в сворачивании на соседнюю полосу в случае обнаружения препятствия и последующем выравнивании на в этой полосе.

Так, автомобиль успешно держит полосу движения на всех протестированных участках дороги, на перекрёстках автомобиль ведёт себя непредсказуемо, так как не может обнаружить линии разметки.

В ходе проведённых экспериментов по обнаружению препятствий было выявлено необходимое количество и расположение датчиков расстояния, а также плотность и количество лучей в этих датчиках. Так, автомобиль способен обнаруживать и объезжать человека, не задевая его боком даже после того, как уже почти объехал его, это решилось выносом области обнаружения датчиков немного вбок, что можно увидеть на рисунке 1.8, а также углом поворота руля при обнаружении препятствия.

1.7 Оценка работы алгоритма с помощью метрик, пути улучшения и выводы

Введём в рассмотрение некоторые показатели, которые помогут понять, насколько «правильно» двигается машина. Во-первых, мы бы хотели, чтоб машина ехала по центру полосы, во-вторых, чтоб она двигалась точно в направлении движения дороги, также было бы неплохо, чтоб она совершала как можно меньше поворотов руля, и не выходила за пределы трассы, не сталкивалась с людьми и другими препятствиями. Для того, чтоб

протестировать и точно оценить последние три пункта, нам потребовалось бы строить достаточно большую и детализированную карту, а результаты не видятся столь интересными, так как нам пока не с чем сопоставить их, и понятно, что этот алгоритм не претендует на применение в широких условиях, поэтому сконцентрируемся на первых двух пунктах.

Для того, чтоб оценить близость машины к центру дороги, необходимо выставить по центру путевые точки, которые камера не будет видеть, но мы сможем на каждом шаге эпизода симуляции найти 2 ближайшие точки, провести между ними отрезок и найти расстояние между этим отрезком и центром автомобиля. Эти данные можно получить из симуляции и записать в лог, а после проанализировать и вычислить среднюю ошибку отклонения от полосы при движении на рассмотренных дорогах (1.7).

$$LDE = \frac{1}{n}\sum_{t=1}^{n} |y_t|, \tag{1.7}$$

где *LDE (Lane Deviation Error)* — средняя ошибка отклонения от центра полосы,

n — общее количество шагов в эпизоде,

$|y_t|$ — абсолютное значение поперечного отклонения центра тяжести автомобиля от центра полосы на шаге t.

Для того, чтобы оценить, насколько движение машины соответствует направлению дороги, можно использовать всё те же путевые точки и оценить угол между отрезком соединяющим две ближайшие путевые точки и фактическим вектором скорости автомобиля. Для удобства оценки можно взять косинус и чем ближе косинус этого угла к единице в среднем по всем шагам эпизода, тем лучше работает алгоритм (1.8).

$$CPA = \frac{1}{n}\sum_{t=1}^{n} cos(\alpha_t) = \frac{1}{n}\sum_{t=1}^{n} \frac{v_t * d_t}{||v_t|| * ||d_t||}, \tag{1.8}$$

где *CPA (Cosine Path Alignment)* — выравнивание траектории машины с дорогой (чем ближе *CPA* к 1, тем лучше выравнивание траектории машины с направлением дороги, значение, близкое к нулю или отрицательное, говорит о сильных отклонениях от курса или движении в обратную сторону),

n — общее количество шагов в эпизоде,

$cos(\alpha_t)$ — косинус угла между отрезком и фактическим вектором скорости автомобиля (чем ближе к 1, тем меньше угол),

v_t — вектор скорости автомобиля на шаге t,

d_t — вектор, построенный от одной ближайшей путевой точки к другой на шаге t,
$||v_t||$ и $||d_t||$ — нормы соответствующих векторов.

В результате эксперимента на показанных на рисунке 1.6 трассах и некоторых других участках были установлены значения LDE и CPA, равные приблизительно 0.15 и 0.9.

Некоторые ограничения данного подхода решения задачи управления направлением движения были отмечены в пунктах 1.5, 1.6, подведём итоги:

Алгоритм способен управлять автомобилем в чётко заданных условиях (определённая скорость, отсутствие других машин, есть куда перестроиться в случае обнаружения препятствий, чёткая разметка, конкретные допустимые углы изгиба дорог и прочее), если попытаться сделать этот алгоритм более универсальным, то придётся прописывать гораздо большее количество условий, возможных углов поворота и прочее, и всё равно не удастся добиться универсальности, а процесс подбора параметров будет очень трудоёмким, имеет смысл вместо этого изменить сам подход принятия решений и изменить процесс тестирования, сделать его более автоматическим (на этом строиться вторая глава работы).

Алгоритм сталкивается с трудностями при плохих погодных условия, потому что не сможет обнаружить линии, на которые ориентируется, поэтому для большей надёжности в принятии решений, а также для детектирования других объектов на дороге можно внедрить более сложные алгоритмы, такие как глубокое обучение.

У алгоритма присутствует сильная перекомпенсация, так как использование локальных правил для корректировки направления движения приводит к чрезмерным поворотам, что снижает стабильность движения.

В следующей главе мы сосредоточимся на использовании методов машинного обучения, которые позволят машине самостоятельно выучить правила для движения на местности, а также улучшать себя при появлении новых условий и дорог несмотря на снег и прочее.

ГЛАВА 2
ИСПОЛЬЗОВАНИЕ ГЛУБОКИХ НЕЙРОННЫХ СЕТЕЙ ДЛЯ УПРАВЛЕНИЯ ДВИЖЕНИЕМ В СЛОЖНЫХ ПОГОДНЫХ УСЛОВИЯХ

2.1 Нейронные сети, принципы их работы и обучения

2.1.1 Однослойные нейронные сети, искусственный нейрон

В главе 1 мы разработали алгоритм на основе нескольких классических методов компьютерного зрения (методы известные до широкого развития нейросетей), классические методы популярны и сейчас, в задачах, в которых использование нейросетей излишне. В этой главе мы рассмотрим более современные подходы, а именно использование моделей на основе нейросетей для управления движением автомобиля.

Рассмотрим для начала базовые понятия, такие как модель, нейронная сеть, искусственный нейрон, перцептрон, обучение модели, функция активации, нейронная сеть, искусственный интеллект, машинной обучение, глубокая нейронная сеть и некоторые другие.

Начнём с понятия модели, модель — это алгоритм, который получает на вход некоторое количество данных, применяет к ним различные линейные и нелинейные функции и предсказывает ответ. Модель может являться нейронной сетью, деревом решений, линейной регрессией. Нейросеть — это вид модели, которая состоит из одного или нескольких слоёв искусственных нейронов. Есть разные виды слоёв, например в полносвязном слое, каждый нейрон слоя связан с каждым нейроном следующего слоя. Изначально, базовым структурным элементом модели являлся перцептрон (вид искусственного нейрона с пороговой функцией активации), перцептрон — это первая относительно успешная попытка реализовать искусственный нейрон, на вход перцептрону по дугам поступают несколько числовых значений, каждой дуге присвоен свой вес, также, для перцептрона задано смещение (2.1). Перцептрон суммирует все поступившие ему значения (учитывая веса) и смещение, а потом применяет к ним пороговую функцию активации и возвращает на выход 0 или 1 (1 если нейрон активировался, иначе 0). Это можно записать так (для простоты пусть поступивших значений было 2):

$$g(x_1, x_2) = \begin{cases} 1, x_1\omega_1 + x_2\omega_2 + b > T, \\ 0, \text{иначе}, \end{cases} \quad (2.1)$$

где g — пороговая функция активации, которую мы применили к перцептрону (результат работы этой функции поступает на выход, подробнее про неё ниже);
$x_1,\ x_2$ — значения, поступившие на вход перцептрону;
$\omega_1,\ \omega_2$ — веса соответствующих дуг, по которым поступают значения;
b — смещение, заданное в перцептроне;
T— пороговое значение, например 0.

Перцептрон представляет собой простейшую нейронную сеть (один слой из одного нейрона). Если представить нейронную сеть в виде графа, то вершинами будут соответствующие функции, которые применяются к поступающим значениям в перцептроны, на вход которым по дугам графа передаются значения, и которые в свою очередь передают результат функции дальше. Таким образом, обучить модель означает вычислить все веса дуг и все смещения для каждого перцептрона.

Можно заметить, что комбинация переменных и постоянных, которая позже сравнивается с порогом, не является линейной в математическом смысле, но при этом в ней нет нелинейных операций таких как возведение в степень, логарифмирование, взятие синуса, при этом входы (признаки) влияют прямо пропорционально своим весам и между ними нет зависимостей, такие функции называются линейными в контексте обучения нейросетей. Представим себе нейронную сеть, которая построена из таких перцептронов. Она сможет моделировать только близкие к линейным зависимости, поэтому она даст прогнозы плохого качества на большинстве практических задач.

2.1.2 Аппроксимация нелинейных функций глубокими нейронными сетями

Чтоб внести нелинейность, появляется необходимость использовать другие функции активации (в задачах распознавания речи, обработки изображений, обработка и анализ текстов). Функция активации называется так, потому что она, как будто, активирует нейрон, в самом простом случае по порогу, то есть если значение вычисленное при суммировании больше некоторого порога, то нейрон активирован и его выход $g(x_1, x_2)$ передаётся на следующий слой нейронной сети (тем искусственным нейронам входы которых соединены с выходом текущего нейрона). В дальнейшем понятие перцептрона обобщилось до искусственного нейрона, у которого в качестве функции активации могут выступать синус, тангенс, ReLu и другие, которые помогают внести в модель необходимую нелинейность. Каждая из функций

активации имеет свои преимущества и недостатки, связанные с процессом обучения нейросетей и типом решаемой задачи [13].

Слои нейронной сети, которые расположены между входом и выходом, называются скрытыми. В [3] была доказана теорема, которая утверждает, что нейронная сеть прямого распространения (без циклов между нейронами) с одним скрытым слоем способна аппроксимировать любую непрерывную функцию с любой точностью при достаточном количестве нейронов в слое и при правильном подборе параметров (веса, смещения).

Однако такой подход не практичен из-за необходимости иметь большое количество нейронов. На практике лучше применим подход, при котором добавляется несколько скрытых слоёв с меньшим количеством нейронов. Так, каждый последующий слой усиливает прогноз, за счёт того, что опирается на прогноз предыдущего слоя, с каждым скрытым слоем глубина сети увеличивается и такие сети называют глубокими (больше 3 слоёв).

2.2.3 Обучение многослойных нейронных сетей

Рассмотрим алгоритм обучения моделей искусственного интеллекта. Нейронные сети, состоящие из нескольких слоёв, обучаются с помощью методов обратного распространения ошибки (backpropagation) и градиентного спуска (gradient descent), алгоритм включает в себя вычисление прогноза, сравнение его с истинным значением и распространение ошибки по сети в обратном направлении пересчитывая при этом веса (делая шаг противоположный направлению градиента). Процесс повторяется до тех пор, пока ошибка не перестанет уменьшаться. Рассмотрим этот процесс подробнее на примере двухслойной сети с двумя нейронами на каждом слое. К нейронам могут быть применимы различные функции активации, пусть в нашем примере будет применена сигмоида. На вход поступает вектор из двух значений, и на выход тоже вектор из двух значений. Рассмотрим для начала архитектуру сети. Входной слой описан формулами (2.2), (2.3).

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, W^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \end{bmatrix}, b^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix}, \quad (2.2)$$

где x — вектор, поступающий на вход,

$W^{(1)}$ — матрица весов первого слоя (скрытого), причём $w_{ij}^{(1)}$ — вес дуги, по которой поступает числовое значение из *j*-ого входа на *i*-ый нейрон перовго слоя,

$\mathrm{b}^{(1)}$ — матрица смещений для нейронов первого слоя.

$$z^{(1)} = W^{(1)}x + b^{(1)},\ a^{(1)} = \sigma(z^{(1)}), \tag{2.3}$$

где $\mathrm{z}^{(1)}$ — вектор размерности 2, первый элемент которого — это сумма взвешенных числовых значений и смещения первого нейрона, второе — для второго нейрона,
$\mathrm{a}^{(1)}$ — вектор размерности 2, который является результатом применения к вектору сумм функции активации (2.4).

$$\sigma(\mathrm{z}) = \frac{1}{1 + \mathrm{e}^{-\mathrm{z}}}, \qquad \sigma'(\mathrm{z}) = \sigma(\mathrm{z})(1 - \sigma(\mathrm{z})), \tag{2.4}$$

где $\sigma(\mathrm{z})$ — функция активации сигмоида, $\sigma'(\mathrm{z})$ — её производная, представленная в удобном для подсчётов виде.

Второй слой выглядит аналогично, но теперь для него входным вектором является не x, а $\mathrm{a}^{(1)}$. Для того, чтоб мы могли вычислить насколько прогноз близок к истине, необходимо ввести функцию, например среднеквадратичное отклонение (2.5):

$$L = \frac{1}{2}\sum_{i=1}^{n}(a_i^{(2)} - y_i)^2, \tag{2.5}$$

где L — функция потерь,
$a_i^{(2)}$ — i-ый элемент вектора, полученного на выходном слое (вектора предсказания),
y_i — i-ый элемент вектора фактического ответа (заданные значения, к которым нужно приблизиться путём обучения),
n — количество выходов.

Изначально веса и смещения инициализируются случайным образом. Алгоритм начинает свою работу с прямого прохода (forward pass). Вычисляется $\mathrm{z}^{(1)}$, $\mathrm{a}^{(1)}$, $\mathrm{z}^{(2)}$, $\mathrm{a}^{(2)}$ по порядку. Далее начинается обратное распространение ошибки и градиентный спуск.

Основная идея заключается в том, что мы хотим уменьшить ошибку, вспомним что производная в случае функции с одним аргументом или градиент для функции с несколькими аргументами показывает направление роста функции.

Рассмотрим это на примере производной функции $f(x)$ в некоторой точке x_0. Ясно, что если в некоторой окрестности точки x_0 производная положительная, то функция в этой окрестности возрастает, и для того, чтобы

уменьшить значение функции, нам нужно сделать маленький шаг влево, не выходя из окрестности, если же производная отрицательная, то функция убывает, и мы должны сделать шаг вправо, чтоб уменьшить функцию. Таким образом, получается, что после шага по направлению обратному производной (градиенту) наша функция уменьшится (2.6).

$$\mathrm{f}(x_0 - l * f'(x_0)) < \mathrm{f}(x_0), \forall\ f'(x_0), \tag{2.6}$$

где l — скорость обучения (обычно устанавливается какое-нибудь маленькое число, например l = 0.01). Если поставить значение слишком большим мы можем бесконечно прыгать вокруг локального минимума, не подходя к нему на достаточно близкое расстояние, а если слишком маленьким, то очень долго сходиться.

Итак, мы разобрались с тем, как именно работает градиентный спуск на примере функции одной переменной, разберёмся, как именно осуществляется обратный проход. Мы выяснили, что нам не так важна сама функция потерь, как её градиент по матрице $W^{(2)}$, вектору $b^{(2)}$ для того, чтобы вычислить новые веса на выходном слое (2.7).

$$W^{(2)} := W^{(2)} - \mathrm{l} * \nabla_{W^{(2)}}\mathrm{L}\quad,\ b^{(2)} := b^{(2)} - \mathrm{l} * \nabla_{b^{(2)}}\mathrm{L} \tag{2.7}$$

На первом слое процесс проходит аналогично.

2.2.4 Преимущества использования нейронных сетей над классическими алгоритмами машинного обучения

Итак, мы рассмотрели алгоритм, по которому обучаются нейронные сети, стремясь минимизировать ошибку в прогнозе. Они обучаются на основе полученных ранее данных делать прогнозы на незнакомых данных, уточним, что же всё-таки включает в себя понятие искусственный интеллект, и отличается ли он чем-то от машинного обучения (если да, то зачем существует два понятия, которые обозначают одно и то же), какие типы задач решает и какие конкретно данные нужны для запуска обучения.

Итак, искусственный интеллект включает в себя всё то, что позволяет решать задачи, требующие человеческого интеллекта, а машинное обучение — это часть искусственного интеллекта, целью которой является научить машину действовать, обучаясь на ранее полученных данных. Машинное обучение включает в себя не только нейросети, но и другие алгоритмы, такие как метод опорных векторов, деревья решений и прочие. Впервые о искусственном интеллекте упомянули, ещё в 1950-ых, когда не существовало

настоящих компьютеров, в то же время была предложена первая модель перцептрона (линейная модель, которая на деле плохо аппроксимирует большинство реальных закономерностей), и из-за чрезмерно оптимистичных прогнозов и инвестиций, вложенных в эту сферу, которые не смогли оправдать однослойные линейные сети, на два десятка лет были остановлены многочисленные исследования по всему миру. С появлением возможности глубже изучить нелинейность, путём добавления большего количества скрытых слоёв, и способностью обучать такие сети методом обратного распространения ошибки, а также благодаря активному развитию аппаратного обеспечения, способного к быстрым операциям над матрицами, с доказательством универсальной аппроксимационной теоремы (которая показала преимущества нейросетей над любым другим подходам), и развитием интернета (источник большого количества данных) нейросети смогли продемонстрировать свою силу и снова стали популярными. Был создан датасет «ImageNet» с 14 миллионами изображений, организован «ImageNet Large Scale Visual Recognition Challenge» (ILSVRC), в котором ежегодно исследователи стремились ко всё более точным прогнозам при распознавании объектов на изображениях. И хотя ещё в 2010 ошибок было почти 30 процентов, к 2012 уже появилась свёрточная сеть, которая дала 16 процентов ошибок, и которая была обучена всего за неделю на двух GPU, а по мере того, как свёрточные сети становились глубже ошибка постоянно улучшалась и уже с 2016 составляла не более 3-ёх процентов.

2.2 Обучение с учителем

2.2.1 Обучение с учителем применительно к задаче автопилотирования автомобиля

Рассмотрим теперь основные виды машинного обучения: обучение с учителем, обучение без учителя и обучение с подкреплением.

В обучении с учителем модель обучается на заранее размеченных данных, решается задача классификации или регрессии. В процессе обучения с учителем не обязательно будут использованы нейросети, это могут быть логистическая регрессия, деревья решений, метод опорных векторов, k-ближайших соседей. Рассмотрим подробнее задачи решаемые методом обучения с учителем. Начнём с задачи классификации. У нас есть ограниченное количество классов, и модель делает прогноз с какой вероятностью объект принадлежит каждому из этих классов. И среди этих вероятностей ищем максимальную, делаем вывод, что модель принадлежит

классу, вероятность для которого максимальная. Рассмотрим некоторые примеры таких задач.

Есть текст письма и модель относит письмо к одной из двух категорий «спам» или «не спам». Нужно распознать объект на фото и модель делает прогноз «кот», «собака» или «не кот и не собака» (3 класса). Алгоритм решения, опять же, не обязательно основан на нейросетях, это может быть метод опорных векторов или наивный байесовский алгоритм, а также другие.

Второй вид задач, решаемых обучением с учителем, — регрессия, суть таких задач предсказать некоторое действительное число. Это может быть предсказание цены или температуры. Такие задачи могут решаться линейной регрессией, регрессионными деревьями, нейронными сетями.

Если посмотреть на этот вид машинного обучения применительно к нашей задаче автопилотирования, то можно выделить несколько подзадач, для решения которых может быть применимо обучение с учителем. Рассмотрим эти подзадачи.

Первое, что приходит на ум, это предсказание углов поворота руля, скорости как действительных значений по данным с датчиков (камера, лидар и прочие). То есть, по сути, это задача регрессии. При решении такой подзадачи мы не планируем пока что распознавать различные объекты на дороге, такие как другие транспортные средства, пешеходы, знаки. Для того чтоб решить такую задачу учитывая описанные выше ограничения (дорога пуста, нет перекрёстков так как не умеем распознавать знаки) при любых условиях (то есть любая дорога, любая разметка или её отсутствие, возможная заснеженность, лужи, туман, дорога имеет уклон или ровная) нам понадобиться огромное количество размеченных данных.

Далее, что можно сделать – это научить модель распознавать объекты (пешеходы, машины, знаки). По сути, это задача классификации. Где на вход модели подаётся изображение, а на выходе мы получаем классы обнаруженных объектов с их координатами и рамками, ограничивающими эти объекты. Для такой задачи хорошо подходят свёрточные нейронные сети.

Можно сегментировать дорогу, то есть присвоить каждому пикселю одну из меток «дорога», «пешеходная часть», «машина», «трава». Это задача пиксельной классификации.

Можно определять намеренья других участников движения, то есть сети подаётся на вход история движения автомобиля на дороге, и она предсказывает собирается ли он ехать прямо, повернуть и т.д.

2.2.2 Сквозное обучение с учителем

Если решать все эти подзадачи по отдельности, то мы получим модульную архитектуру. Это означает, что вся система делится на независимые компоненты, каждый из которых решает свою подзадачу: обнаружение объектов, сегментация сцены, определение дорожной разметки, планирование пути, контроль движения. Каждый модуль при этом основан на своих алгоритмах (не всегда используется машинное обучение), имеет свои входные и выходные данные, которые посредством вручную написанных интерфейсов связывается с другими модулями. Проблема такого подхода заключается в том, что сложно учесть все закономерности, сложно модифицировать интерфейсы и они очень хрупки, а разметка трудозатратна.

Современный подход — это, так называемое, сквозное обучение (end-to-end). Где у нас есть всего один мощный блок, на вход которому поступают сырые данные (пиксели изображения, данные лидаров, GPS), а на выходе мы получаем управляющие сигналы (положение педали газа, поворот руля, траектория). Для того, чтоб обучить такую сеть в обучении с учителем, требуется собрать реальные данные вождения (водитель управляет автомобилем, а все данные с датчиков и управляющие действия записываются). Проанализируем преимущества и недостатки сквозного обучения.

Из преимуществ можно выделить отсутствие необходимости вручную размечать промежуточные признаки (автомобили, линии разметки, прочее), не накапливаются ошибки между модулями, модель может самостоятельно обучиться сложным зависимостям, которые человек бы не смог описать при конструировании интерфейсов, модель гораздо проще разрабатывать.

Конечно, такой подход не лишён и недостатков, при всей своей эффективности. Можно начать с типичной проблемы для глубоких нейронных сетей «чёрного ящика». Не понятно почему модель принимает именно такое решение, трудно адаптировать поведение модели к непонятным ситуациям и новым условиям.

2.2.3 Гибридное обучение с учителем

Как компромисс между, понятным логически, разделением задачи на модули и «чёрным ящиком» сквозного обучения, существует, так называемый, гибридный подход, который используют лидеры рынка беспилотных автомобилей. Рассмотрим на примере Tesla.

Согласно [8], [2], [16], [17], Tesla используют глубокие нейросети, обрабатывающие пиксели с камер прямо до управляющих действий, но с

архитектурой, где модули обучаются отдельно, а работают как единая сеть. Компания использует модули, но они связаны напрямую, то есть выходы первого модуля прямо поступают на вход следующему модулю, без прописанной вручную интерфейсной логики. Так в их системе автопилота Full Self-Driving (FSD) начиная от пикселей до предсказания управляющих сигналов нет ручных правил или ручной сегментации — всё учится через данные. Рассмотрим подробнее систему FSD, которая включает в себя три основных блока. Первый блок — блок восприятия, он обрабатывает данные с восьми камер, создавая представление окружающей среды, этот блок использует нейросети для идентификации объектов, дорожных знаков и разметки. Данные из этого слоя напрямую передаются во второй модуль — модуль модели мира в качестве многомерного тензорного представления — карты признаков, 2D/3D рамок, ограничивающих объекты, сигналов наличия объектов, их классов, вероятностей.

Модель мира формирует трёхмерное представление окружающей среды, включая статические и динамические объекты, определяет, какие области пространства заняты / свободны, прогнозирует будущие положения объектов. 3D-модель обучается из видео, без использования лидаров (датчики расстояния), она учится понимать не просто "что есть", а "где это находится", "как это движется", "что с этим будет дальше".

Модуль планирования на основе модели мира определяет оптимальную траекторию движения (учитывает правила дорожного движения, поведение других участников и прогнозирует возможные сценарии развития ситуации).

Эти компоненты тесно интегрированы, обеспечивая слаженную работу системы FSD, такая архитектура позволяет интерпретировать решения путём анализа выходных данных каждого модуля, понимать, как конкретные входные данные влияют на конечные решения, обеспечивать возможность отладки и улучшения отдельных компонентов без необходимости пересмотра всей системы.

2.2.4 Проблема данных при обучении автопилотированию с учителем

Как уже было сказано ранее, для обучения с учителем необходимо огромное количество обучающих данных. Рассмотрим исследование, проведённое RAND Corporation [7] в 2016 году. Целью исследования было оценить сколько километров необходимо проехать автономным автомобилям, чтобы статистически доказать свою безопасность по сравнению с водителями-людьми, по трем сценариям: безаварийная езда (сколько км без аварий нужно проехать, чтобы с уверенностью заявить о

низком уровне отказов), определённая точность (сколько км нужно проехать, чтобы оценить уровень отказов с заданной точностью), сравнение с людьми (сколько км нужно, чтобы доказать, что автопилот лучше человека по уровню безопасности). Итак, чтобы доказать, что автопилот безопасен на уровне не хуже 1.09 смертей на 160 млн км, нужно 442 млн км без единого смертельного случая. Чтобы точно оценить фактический уровень смертности автопилота, нужно 14.2 млрд км. Чтобы доказать, что автопилот на 20% безопаснее (имеет уровень 0.872 смертей на 160 млн км) требуется 18.2 млрд км. Но парадокс заключается в том, что важен не просто пробег, а реальные аварийные ситуации, и в реальном мире, если система безопасная, то такие ситуации случаются редко. То есть чем более безопасная система, тем труднее доказать её безопасность в аварийных, нестандартных ситуациях. Посмотрим на примере Tesla, по состоянию на 2024 год у них примерно 5 миллионов автомобилей, которые ежегодно проезжают более 80 миллиардов километров. Несмотря на это, компания всё ещё не может собрать достаточно данных в реальном мире. В реальном мире аварии, почти-аварии, странные ситуации — редки. Даже с миллиардами миль, нужных именно "краевых случаев" — мало. А нейросеть должна быть устойчива именно к таким случаям (ребёнок выбегает из-за машины, мотоциклист едет между рядами, странная разметка). Симуляции позволяют создавать миллионы вариантов одной опасной сцены, модифицировать её и изучать, как поведение модели меняется при небольших изменениях. Tesla использует нейросети, обучаемые не только на реальных, но и на синтетических данных (например, в сценах, где данные из реального мира невозможно получить без риска). Например, в 2023 году Tesla начала активно использовать суперкомпьютер Dojo, который может запускать миллионы сцен для тренировки сетей.

Можно сделать вывод, что реальные поездки не позволят статистически доказать безопасность автопилота по сравнению с человеком, особенно если автопилот уже близок к человеческому уровню, это влечёт необходимость использовать симуляции, в которых помимо прочего моделировать редкие и опасные сценарии, чтоб получать именно полезные для обучения данные, требуется проехать те же миллиарды километров, но в симуляциях, учитывая краевые случаи.

2.3 Обучение без учителя

Обучение без учителя — это вид машинного обучения, в котором модель обучается на данных без заранее заданных примеров, на основе которых она должна делать предсказания. В отличие от обучения с учителем,

где алгоритм получает пары «вход-выход» и учится на основе этих данных, обучение без учителя использует только входные данные и пытается выявить скрытые структуры или закономерности. Рассмотрим основные задачи, которые решает обучение без учителя, и как можно сформулировать эти задачи для автопилотируемых автомобилей.

Начнём с кластеризации, задача кластеризации — это задача разделения объектов на группы, в каждой группе содержаться похожие друг на друга объекты. Кластеризация может быть выполнена методами k-средних, Gaussian Mixture Models (см. [12], [22]), которые позволяют учитывать выбросы и масштабируемость при работе с большими объёмами данных. Применительно к нашей задаче, кластеризация может использоваться для разделения потока точек на кластеры, где в одном кластере будут содержаться объекты похожие на автомобили, в другом, похожие на пешеходов, и т. д. Кластеризация также может помочь отделить элементы дороги друг от друга (проезжую часть отделить от обочины и препятствий). кроме того, можно использовать её для предобработки данных (например, структурировать их, чтоб упростить их использование в обучении с учителем).

Далее, рассмотрим задачу понижение размерности. По сути, благодаря понижению размерности мы можем визуализировать многомерные данные или сжимать изображения или облака точек лидара перед последующей обработкой. Методы решения задачи понижения размерности включают в себя линейное снижение размерности, нелинейные t-SNE / UMAP (для визуализации обучающих признаков), или автоэнкодеры (нейросетевые методы для сжатия и восстановления данных с сохранением структуры), способные выделять наиболее значимые признаки в высокоразмерных данных [5].

Одной из важных задач обучения без учителя является задача обнаружения аномалий. Это задача обнаружения необычного поведения других участников движения (например, пешеход выбежал на дорогу), распознавания неисправностей сенсоров или программных модулей. Для решения используют методы изоляционного леса, одноклассовый метод опорных векторов, эффективность которых была сравнительно изучена в работе [20].

Посмотрим на реальные примеры использования обучения без учителя лидерами рынка автономных автомобилей. Например, компания Waymo разработала подход [6], который позволяет системе самостоятельно обучаться на неразмеченных данных, используя информацию о движении объектов. В подходе модули восприятия и предсказания поведения в системе автопилота обучаются с использованием обучения без учителя.

Система анализирует последовательности данных с лидаров, чтобы выявить движущиеся объекты и отделить их от статического фона. Используя информацию о движении, система автоматически создаёт метки для объектов, которые затем используются для обучения моделей распознавания и предсказания без участия человека.

В отличие от традиционных моделей, ограниченных заранее определёнными категориями объектов, предложенный метод способен распознавать и предсказывать поведение ранее неизвестных объектов, что особенно важно для безопасности на дорогах.

Для этого метода, были проведены эксперименты на открытом датасете Waymo Open Dataset, которые показали, что он значительно превосходит классические методы обучения без учителя в задачах 3D-детекции объектов, достигает результатов, сопоставимых с моделями, обученными с учителем, при этом не требуя ручной разметки данных. Этот подход позволяет системе эффективно обучаться на больших объёмах неразмеченных данных, что снижает зависимость от дорогостоящей ручной разметки, адаптироваться к новым и ранее неизвестным объектам и ситуациям на дорогах, повысить общую безопасность и надёжность системы автономного вождения.

2.4 Обучение с подкреплением

Перейдём к обучению с подкреплением (reinforcement learning, RL). Обучение с подкреплением — это вид машинного обучения, при котором агент взаимодействует с окружающей средой и получает в результате своих действий вознаграждения за правильные действия и штрафы за неправильные. Ключевым отличием от обучения с учителем является то, что от агента не требуется имитировать действия человека по полученным примерам, от него требуется методом проб и ошибок научиться самостоятельно принимать «хорошие» решения, то есть, потенциально он может открыть такие стратегии поведения, о которых человек бы и не подумал, поскольку его обучение не ограничено заранее размеченными примерами, а строится на поиске оптимальной политики действий, максимизирующей долгосрочную награду. В отличие от обучения с учителем, где модель учится на "правильных" ответах, RL-агент сам оценивает последствия своих действий во времени. Награда может быть отложенной — результат действия проявляется через десятки или сотни шагов, и это делает обучение особенно сложным, но и мощным. Именно поэтому обучение с подкреплением так успешно применяется в задачах, требующих долгосрочного планирования, стратегического мышления и адаптации таких как го и шахматы.

В 2022 году агент Diplodocus, обученный с использованием RL и планирования, занял первые места в турнире по игре Diplomacy среди 62 участников разного уровня, включая новичков и экспертов. Агент продемонстрировал стратегическое мышление и способность к сотрудничеству, что позволило ему превзойти человеческих игроков [16]. В 2023 году агент, обученный с использованием иерархического обучения с подкреплением (HRL), продемонстрировал способность адаптироваться к различным стилям игры и уровням навыков реальных игроков в игре Overcooked. Агент эффективно сотрудничал с людьми, улучшая общую производительность команды [20]. В 2024 году агент RAISocketAI, обученный с использованием глубокого обучения с подкреплением (DRL), стал первым агентом, победившим в соревновании microRTS, обыграв предыдущих победителей. Агент продемонстрировал стратегическое планирование и адаптацию к различным игровым ситуациям [18]. В 2025 году компания Figure использовала обучение с подкреплением для улучшения походки своих гуманоидных роботов, сделав их движения более естественными. Роботы были протестированы в реальных условиях, включая установку BMW, и продемонстрировали способность эффективно взаимодействовать с людьми [19].

Рассмотрим основные элементы, на которых строиться обучение с подкреплением: агент, окружение, состояние, действие, награда, политика, функция ценности.

Агент — это сущность, которая принимает решения в процессе взаимодействия с окружением. Агент может быть, например, роботом, автономным автомобилем или программным обеспечением, принимающим решения. Окружение может быть как физическим, так и виртуальным. Например, в задаче с игрой — это сама игровая среда, в задаче с роботами — физический мир, а в задаче с оптимизацией — математическая модель. Состояние — это полная информация о текущем моменте взаимодействия агента с окружением. Это может быть вектор, представляющий положение робота, или состояние игры. Действие — это выбор, который агент делает в конкретном состоянии. Например, если агент управляет роботом, его действия могут быть «вперёд», «влево», «вправо» или «остановиться».

Вознаграждение — это числовое значение, которое агент получает после выполнения действия. Оно отражает степень успешности действия в текущем состоянии. Цель агента — максимизировать суммарное вознаграждение за весь период обучения.

Политика — это стратегия или правило, которое определяет, какие действия агент должен предпринимать в каждом состоянии. Политика может быть детерминированной или стохастической.

Функция ценности — это функция, которая оценивает, насколько выгодно находиться в определённом состоянии или предпринимать определённое действие, с учётом будущих вознаграждений.

Цель задачи обучения с подкреплением — это нахождение оптимальной стратегии (политики), которая максимизирует суммарное вознаграждение агента в течение времени. Это может быть достигнуто через методы, такие как Q-обучение, глубокое обучение с подкреплением и другие. В [9], [14] очень подробно рассмотрена механика обучения с подкреплением, выделим некоторые ключевые моменты и проанализируем их.

2.4.1 Марковский процесс принятия решений как основа обучения с подкреплением

Марковский процесс принятия решений (MDP) — это математическая модель, используемая для описания задач последовательного принятия решений. Формально MDP задаётся пятёркой (2.8).

$$(\mathcal{S}, \mathcal{A}, P, R, \gamma), \tag{2.8}$$

где $\mathcal{S}$ — множество состояний среды,
$\mathcal{A}$ — множество действий,
P — функция вероятности перехода в следующее состояние,
R — функция вознаграждения,
$\gamma \in [0,1]$ — коэффициент дисконтирования, определяющий важность будущих наград.

С множеством состояний и действий всё понятно, рассмотрим остальные элементы. Начнём с функции вероятности перехода (2.9).

$$\mathrm{P}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1], \tag{2.9}$$

где $\mathcal{S}$ — множество всех возможных состояний,
$\mathcal{A}$ — множество всех возможных действий агента.

То есть, у нас есть для каждого действия своя матрица перехода между состояниями размера $n \times n$, где n – количество состояний, строки соответствуют текущему состоянию s, а столбцы — следующему состоянию s'. Тогда можно посчитать вероятность попадания в конкретное состояние на следующем шаге (2.10) которое зависит только от текущего состояния и действия.

$$P(s'|s,a) = \mathbb{P}(s_{t+1} = s' \mid s_t = s, a_t = a\ ), \tag{2.10}$$

где $P(s'|s,a)$— условная вероятность того, что система попадёт в состояние s' на следующем шаге времени $t+1$, при условии, что на текущем шаге времени агент находится в состоянии s и выполняет действие a,
s_t — состояние среды в момент времени t,
a_t — действие агента, выбранное в момент времени t,
s_{t+1} — состояние среды в следующий момент времени,
$\mathbb{P}$ — вероятность, определяемая вероятностной моделью среды.

Формула (2.10)показывает ключевое свойство Марковского процесса: независимость будущего состояния от всей истории действий/состояний целиком, а только от предыдущих. Так, выполняется (2.11).

$$\mathbb{P}(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \dots) = \mathbb{P}(s_{t+1} \mid s_t, a_t), \tag{2.11}$$

где s_{t-1}, a_{t-1} — состояния и действия, которые нет необходимости учитывать.

Перейдём к рассмотрению функции вознаграждения R, она ставит паре состояния и предпринимаемого действия из этого состояния некоторую оценку, которая является действительным числом (2.12).

$$R: \mathcal{S} \times \mathcal{A} \to \mathbb{R}, \; R(s,a) = \mathbb{E}[r_{t+1} \mid s_t = s, a_t = a], \tag{2.12}$$

где R — функция вознаграждения, принимающая в качестве значений действительные числа ($\mathbb{R}$), заданная на декартовом произведении $\mathcal{S} \times \mathcal{A}$,
r_{t+1} — вознаграждение, которое агент получает после выполнения действия,
$\mathbb{E}$ — математическое ожидание (среднее значение по всем возможным результатам,
$R(s,a)$ — средняя награда за выполнение действия a из состояния s.

Теперь оценим роль коэффициента дисконтирования γ, как было сказано ранее он задаётся числом от 0 до 1 и определяет важность будущих наград. То есть, он задаёт, насколько сильно агент ценит будущие вознаграждения по сравнению с текущими. Когда мы суммируем награды во времени (по всем кадрам в эпизоде), мы используем дисконтированную сумму (2.13).

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1}, \tag{2.13}$$

где G_t— суммарное вознаграждение начиная с момента *t* (кадра под номером *t* и до конца эпизода),

r_{t+k+1} — вознаграждение, полученное на шаге *t+k+1,*

γ^k— вес (важность) этого вознаграждения (уменьшается).

Таким образом, если $\gamma = 0$, то агент видит только текущую награду (жадный выбор). Если $\gamma \approx 0.9 - 0.99$, то агент оптимизирует долгосрочные награды. Чем ближе γ к 1, тем дальше агент смотрит в будущее, тем больше он заботится о будущей выгоде.

Перейдём к формализации политики, понятно, что цель агента — научиться выбирать оптимальную стратегию действий (политику), которая рассчитывается по формуле (2.14).

$$\pi^* = arg\ \max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t\, r_{t+1} \right], \tag{2.14}$$

где π^* — оптимальная стратегия (та, которая максимизирует ожидаемое вознаграждение за весь эпизод),

π — стратегия (политика) агента (правило, по которому он выбирает действия в зависимости от состояния),

argmax — поиск среди политик π той, которая максимизирует выражение,

$\mathbb{E}_{\pi}$ — математическое ожидание награды, при следовании стратегии π.

Формула (2.14) подразумевает, что будет найдена такая стратегия π^*, при которой ожидаемая суммарная дисконтированная награда максимальна.

Для каждой политики удобно ввести функцию ценности (2.15), которая будет оценивать не просто награду за кадр, как это делает функция вознаграждения, а суммарную дисконтированную награду за все кадры эпизода (то есть функция ценности просчитывает далёкое будущее, а не просто оценку на следующем шаге). То есть функция ценности — это математическая форма оценки того, насколько "выгодно" следовать политике π (выполнять те или иные действия) из заданного состояния (в долгосрочной перспективе).

$$V^{\pi}(s) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t\, r_{t+1} \mid s_0 = s \right], \tag{2.15}$$

где $V^{\pi}(s)$ — среднее ожидаемое вознаграждение, если стартовать из состояния s и следовать стратегии π,

$\mathbb{E}_{\pi}$ — математическое ожидание награды, при следовании стратегии π,

s_0 — начальное состояние.

По аналогии с функцией ценности (2.15) можно ввести функцию ценности действия (2.16), которая в отличие от (2.15) учитывает ещё и первое действие.

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t}\, r_{t+1} \mid s_0 = s, a_0 = a\right], \tag{2.16}$$

где $Q^{\pi}(s,a)$ — ожидаемая награда при старте в s, выполнении действия a и дальнейшем следовании π,

a_0 — первое действие.

Мы рассмотрели основные элементы марковского процесса принятия решений, проанализируем, почему именно марковские процессы так необходимы в обучении с подкреплением.

Вспомним, какую проблему решает RL. Обучение с подкреплением — это про обучение агента на основе опыта взаимодействия с окружающей средой: агент не знает заранее правил среды, он получает награды/штрафы за свои действия, он должен выработать стратегию, которая максимизирует суммарную награду. И именно для того, чтоб описать такую ситуацию формально и точно необходима математическая модель, которая учитывает случайность среды, описывает взаимосвязь состояний, действий и наград, позволяет говорить об оптимальности поведения агента. Без MDP мы бы не смогли чётко и строго определить задачу обучения: что означает «хорошее поведение», как описать процесс принятия решений, как обобщать поведение на будущее? Принцип «будущее зависит только от текущего состояния и действия» упрощает расчёты так как не нужно помнить всю историю. Без этого агенту пришлось бы учитывать всю последовательность действий и состояний в прошлом. Это усложняет задачу и делает её невычислимой. Структура MDP задаёт чёткое множество состояний, множество действий, вероятности переходов между состояниями при выполнении каждого действия, функцию награды. Всё это позволяет описать среду как систему уравнений, которую можно изучать, анализировать и использовать для оптимизации.

Перейдём к рассмотрению подходов в обучении с подкреплением.

2.4.2 Подходы в обучении с подкреплением

Обучение с подкреплением можно разделить на два основных типа: обучение на основе политики и обучение на основе оценки [15].

Рассмотрим обучение на основе политики. Здесь, агент напрямую учит себя, как действовать в различных ситуациях. Он не строит картину ценности для каждого состояния, а просто пытается научиться правильной стратегии, которая даст ему наибольшее вознаграждение.

Рассмотрим пример. Агент учится играть в игру и обновляет свою стратегию (политику), чтобы выбирать действия, которые в долгосрочной перспективе приносят большее вознаграждение. Вместо того, чтобы сначала оценивать все возможные состояния (как в оценке), агент сразу пробует действовать в соответствии с текущей стратегией. Пример такого метода служит PPO — это алгоритм, который обновляет стратегию (политику) с учётом того, как хорошо она себя проявила.

Второй вид — это обучение на основе оценки. Агент сначала оценивает, насколько выгодно быть в каждом состоянии или выполнять определенные действия. То есть он пытается понять, насколько ценным будет выполнение того или иного действия в конкретной ситуации.

Рассмотрим предыдущий пример. Агент играет в ту же игру, но сначала он оценивает каждое возможное действие и состояние, пытаясь понять, какое из них принесёт наибольшее вознаграждение в долгосрочной перспективе. Это помогает ему выбрать лучшие действия на основе этих оценок. Примером применения метода служит Q-обучение. В нем агент строит таблицу оценок для каждого состояния и действия (например, Q-значение), и затем выбирает действие с наибольшим значением.

Главное различие: обучение на основе политики — агент учит стратегию, то есть, как ему лучше всего действовать в каждой ситуации, обучение на основе оценки— агент учит, насколько ценными являются разные состояния и — действия, а затем выбирает те, которые кажутся ему лучшими.

Эти два подхода можно комбинировать. Например, в методах Actor-Critic используется и обучение на основе политики (Actor), и обучение на основе оценки (Critic).

2.4.3. Простейшая архитектура сетей Актёра и критика

Рассмотрим простую реализацию нейросетей актёра и критика, о которой говорилось в предыдущем пункте. Целью сети актёра является аппроксимация стохастической политики (2.17).

$$\pi_\theta(a_t|s_t) \approx \mathbb{P}(a_t \mid s_t; \theta), \tag{2.17}$$

где θ — параметры сети (веса и смещения).

Входное пространство $\mathcal{S} \subseteq \mathbb{R}^{\mathrm{n}}$, где n — размерность вектора состояния среды. Выходное пространство: $\mathcal{A} \subseteq \{1,2,\ldots,\mathrm{k}\}$, где k — количество дискретных действий.

Структура сети состоит из входного слоя ($\mathrm{x} \in \mathbb{R}^{\mathrm{n}}$), первого скрытого слоя (2.18):

$$h_1 = ReLU(W_1 x + b_1), \tag{2.18}$$

где $W_1 \in \mathbb{R}^{10\times n}, \mathrm{b}_1 \in \mathbb{R}^{10}$.

Второй скрытый слой (2.19):

$$h_2 = ReLU(W_2 h_1 + b_2), \tag{2.19}$$

где $W_2 \in \mathbb{R}^{10\times 10}$.

И выходной слой (2.20):

$$\mathrm{z} = W_3 \mathrm{h}_2 + \mathrm{b}_3 \in \mathbb{R}^k \tag{2.20}$$

На выходном слое применяем функцию активации софтмакс (2.21):

$$\pi_\theta(a \mid s) = \mathrm{Softmax}(\mathbf{z}) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \tag{2.21}$$

Сеть оценки выглядит очень похожим образом, но поскольку её цель – это аппроксимация ценностной функции состояния (2.22), то выходной слой отличается.

$$V^\pi(s_t) \approx V(s_t;\phi), \tag{2.22}$$

где ϕ — параметры модели критика.

Выходное пространство представляет собой скаляр $V(s) \in \mathbb{R}$, то есть он не содержит активационной функции (так как предсказывается действительное число — оценку полезности состояния)

Подведём итоги. Обе сети имеют идентичную архитектуру скрытых слоёв (по 2 слоя с 10 нейронами каждый), но различаются функцией активации на выходе и размерностью выходного слоя (k – для актёра, 1 – для критика). Такой подход обеспечивает раздельное обучение политики и ценностной функции.

Так, мы рассмотрели в главе 2 различные подходы из машинного обучения которые можно применить для поставленной нами задачи,

проанализировали преимущества и недостатки каждого из них. Следующая глава будет посвящена разработке системы автопилота основанной преимущественно на обучении с подкреплением, поскольку как мы выяснили, этот тип обучения не требует большого количества подготовленных данных и позволяет автомобилю самостоятельно обучаться методом проб и ошибок, кроме того, мы можем чётко задавать, чему конкретно должен научиться автомобиль корректируя функцию вознаграждения.

ГЛАВА 3
ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ ДЛЯ УПРАВЛЕНИЯ ДВИЖЕНИЕМ

3.1 Постановка задачи

Вспомним постановку задачи из пункта 1.1 и два подхода которые мы формулировали для её решения, в этой главе рассмотрим второй подход, а именно, использование нейронных сетей для выбора направления движения. При реализации этого подхода мы хотим не просто решить такую же задачу, а сделать так, чтоб это решение можно было легко обобщить на любую другую дорогу, на дороги с движением автомобилей, на дороги имеющие препятствия, на дороги со сложными погодными условиями и прочее.

В главе 2 мы рассмотрели различные виды машинного обучения: обучение с учителем, без учителя, с подкреплением. Обучать автомобиль мы планируем в симуляторе, и для того, чтоб обобщить построенную модель на более сложные условия (пешеходы, поток транспорта), хотелось бы избежать того, чтоб нам нужно было каждый раз проезжать в симуляторе эти участки самостоятельно управляя автомобилем, либо используя для этого уже готовые автопилоты в качестве учителей, поэтому нам не очень хочется иметь дело с обучением с учителем для такой задачи. Обучение без учителя тоже требует большого количества (хоть и не размеченных) данных. Можно было бы взять уже обученные модели, убрать несколько последних слоёв и дообучить их для работы в симуляторе, что потребует меньшего количества данных, но тут появляются всё те же проблемы, что мы не можем обучить модель без дополнительного сбора данных, и для новых условий нам придётся опять самостоятельно их собирать, а модель не сможет выучить ничего сама, не повторяя поведение водителя. Поэтому для этой задачи было решено выбрать обучение с подкреплением. Именно оно сможет позволить нам, задав соответствующие функции вознаграждения и углубив нейронные сети в методе обучения, быстро и автоматически обобщать модель для работы во все более сложных условиях.

Но, прежде чем заниматься обобщениями, нужно построить каркас, на который их можно накладывать, таким каркасом будет реализованный алгоритм обучения с подкреплением для задачи удержания полосы на однополосной дороге с любой разметкой или без неё, без перекрёстков и прочего, без препятствий на дороге.

3.2 Управление автомобилем с помощью обучения с подкреплением на основе DeepBots

Рассмотрим реализацию интеллектуального управления автомобилем, реализованную нами в симуляторе Webots с использованием обучения с подкреплением. В качестве основного инструмента применяется фреймворк DeepBots, предоставляющий мост между симуляцией Webots и алгоритмами обучения агента на Python. Управление автомобилем осуществляется за счёт обучения агента, взаимодействующего со средой, при этом агент учится принимать оптимальные решения на основе данных с фронтальной камеры, установленных на транспортном средстве.

Рассмотрим, как реализован механизм(цикл) обучения с подкреплением в DeepBots. Архитектура DeepBots построена на взаимодействии двух классов контроллеров: контроллер среды — класс-наследник «CSVSupervisorEnv», контроллер агента — класс-наследник «CSVRobot». Эти два класса представляют разделение обязанностей между средой, отвечающей за логику симуляции и выдачу награды и агентом, отвечающим за обработку наблюдения и выбор действий.

Рассмотрим подробнее контроллер среды. Контроллер среды реализует «объективную точку зрения» на симуляцию. Основные функции: выдача наблюдения для агента, применение действия к роботу (например, изменение угла поворота руля), вычисление награды за текущее поведение агента, определение условия завершения эпизода, периодическое сохранение модели и контроль перехода между режимами обучения и тестирования.

Контроллер агента запускается внутри робота в Webots. Он получает данные с сенсоров (в частности, изображения с фронтальной камеры), выполняет предобработку наблюдения (изменение размера, нормализация и т.д.), передаёт это наблюдение в обучающую систему Python (через файл или сокет), получает действие от модели (например, повернуть руль налево), применяет соответствующее действие к роботу в симуляции.

Обмен между двумя контроллерами в DeepBots реализован через временные CSV-файлы, где происходит передача информации в формате csv: «observations.csv» — текущие наблюдения среды (например, обработанное изображение); «actions.csv» — действия, рассчитанные агентом; «rewards.csv» — значение награды; «dones.csv» — флаг окончания эпизода.

Процесс обмена (цикл обучения) можно описать следующим образом:

1) Контроллер среды фиксирует состояние симуляции, вычисляет награду и определяет завершение эпизода. Контроллер среды записывает наблюдение в «observations.csv».

2) Контроллер агента считывает наблюдение, обрабатывает его и подаёт на вход нейросети.

3) Агент выбирает действие, которое записывается в «actions.csv».

4) Контроллер среды считывает действие и применяет его в симуляции.

5) Цикл повторяется.

Этот механизм позволяет чётко отделить симуляцию от логики обучения, обеспечивая модульность и масштабируемость архитектуры.

Как было упомянуто ранее, использование DeepBots и обучения с подкреплением даёт ряд ключевых преимуществ по сравнению с традиционными методами обучения. Агент самостоятельно обучается принимать решения, используя только данные с камеры и сигналы награды. Это избавляет от необходимости вручную проектировать алгоритмы обработки изображений или поведения. Один и тот же агент может быть повторно использован для дорог с другим радиусом кривизны, многополосного движения, включения других участников движения и объектов. Благодаря обучению на симуляции с разнообразными начальными условиями агент обучается обобщать и реагировать на различные сценарии, даже те, которые не были явно запрограммированы.

Из пункта 2.4 мы знаем, что для обучения с подкреплением важно внимательно сконструировать функцию наград, чтоб агент понимал, какое поведение мы от него ожидаем. Следующий пункт будет посвящён разработанной нами функции вознаграждения и методу её подсчёта.

3.3 Реализация функции вознаграждения

Функция вознаграждения — это ключевой компонент в обучении с подкреплением, определяющий поведение агента, нам важно, создать метрику, которая стимулирует движение по центру полосы и вдоль её направления, а также штрафует за отклонения и выезд с дороги. Мы реализовали, можно сказать, классическую, геометрически осмысленную функцию вознаграждения для движения по одной полосе, которая основана на трёх показателях: расстояние автомобиля до центральной линии дороги, угол между вектором скорости автомобиля и направлением дороги, наличие выхода за пределы допустимой полосы движения. В результате экспериментов мы подобрали подходящие веса для этих показателей, чтоб правильно объяснить модели желаемое поведение. Для того чтоб отслеживать центральную полосу используем путевые точки. Эти точки, задающие геометрию дороги, размещаются вдоль всего маршрута и невидимы для агента — то есть он не получает к ним прямой доступ, что важно для соблюдения реализма.

Чтоб вычислить местоположение автомобиля относительно центральной линии дороги для расчёта награды используем следующий несложный алгоритм.

Для начала нам необходимо найти две ближайшие путевые точки, между которыми автомобиль предположительно находится. Пусть:

$W = \{w_1, w_2, \dots, w_n\} \subset \mathbb{R}^2$ — список путевых точек, а $\vec{p} \in \mathbb{R}^2$ — текущая позиция автомобиля. Тогда определим евклидовы расстояния от автомобиля до каждой точки (3.1):

$$d_i = |\vec{p} - w_i|, \qquad i = 1, \dots, n, \tag{3.1}$$

где d_i — индексы точек.

Затем выбираем два индекса с минимальными значениями d_i. Они определяют отрезок дороги, вдоль которого должен двигаться автомобиль.

Далее нам нужно найти расстояние (3.2) от центра автомобиля до отрезка между двумя ближайшими путевыми точками. Для оценки того, насколько далеко автомобиль отклонился от дороги, рассчитаем расстояние от точки $\vec{p}$ до отрезка $\overline{w_1 w_2}$. Используется формула проекции точки на отрезок. Пусть $\vec{v} = \vec{w}_2 - \vec{w}_1$ — вектор дороги, $\vec{w} = \vec{p} - \vec{w}_1$ — вектор до автомобиля. Найдём скалярное произведение: $c_1 = \vec{w} \cdot \vec{v},\ c_2 = \vec{v} \cdot \vec{v}$.

$$d = \begin{cases} |\vec{p} - \vec{w}_1|, & c_1 \le 0, \\ |\vec{p} - \vec{w}_2|, & c_1 > 0, c_1 \ge c_2, \\ \left|\vec{p} - \vec{p}_{\text{proj}}\right|, \text{где } b = \dfrac{c_1}{c_2}, & \vec{p}_{\text{proj}} = \vec{w}_1 + b \cdot \vec{v}, \quad \text{иначе}, \end{cases} \tag{3.2}$$

где d — расстояние.

Продолжим, чтобы агент двигался вдоль дороги, а не поперёк или назад, используется оценка угла между вектором скорости и вектором дороги. Обозначим $\vec{v}_{\text{road}} = \vec{w}_2 - \vec{w}_1$, $\vec{v}_{\text{car}}$ — проекция вектора скорости на плоскость (ось X и Y), что, по сути, даёт нам направление движения автомобиля. Тогда угол между ними равен (3.3):

$$\theta = \arccos\left(\frac{\vec{v}_{\text{road}} \cdot \vec{v}_{\text{car}}}{|\vec{v}_{\text{road}}| \cdot |\vec{v}_{\text{car}}|}\right), \tag{3.3}$$

где θ — искомый угол,

$|\vec{v}_{\text{road}}|$ — норма соответствующего вектора.

Итак, после получения всех данных можно переходить к расчёту вознаграждения (3.4).

$$R = \phi_1 cos(\delta) - \phi_2 |\frac{P}{W}|, \quad (3.4)$$

где δ — угол отклонения направления движения автомобиля от направления дороги,
P — отклонение машины от центра полосы,
W — ширина дороги,
ϕ_i — неотрицательные весовые коэфиценты.

Поясним немного формулу (3.4). По сути, мы хотим выполнения двух условий: во-первых, нам нужно чтоб машина ехала в правильном направлении, для этого используется первая часть функции вознаграждения. Мы рассчитываем отклонение направления движения от оптимального и берём косинус, чтоб нормировать это значение от 0 до 1 (если отклонение близко к 0, значение близко к 1, чем больше отклонение, тем ближе к нулю). Вторая (отрицательная) часть вознаграждения связана с тем, что мы хотим, чтоб автомобиль двигался по центру дороги, мы нормируем расстояние от центра по ширине дороги для удобства.

Данный подход позволяет агенту формировать устойчивую стратегию движения, максимально приближенную к безопасному стилю вождения.

3.4 Реализация остановки эпизода

При разработке функции завершения эпизода важным решением стало внедрение параметра «max_out_of_road_steps», который определяет допустимое количество последовательных шагов, в течение которых агент может находиться вне пределов дорожной полосы, прежде чем эпизод будет принудительно завершён. Такой подход позволяет не прерывать эпизод сразу при первом нарушении, а дать агенту возможность «осознать» свою ошибку и попытаться её исправить.

Если завершать эпизод немедленно при каждом выезде за границы трассы, агенту будет сложно установить причинно-следственные связи между своими действиями и негативным исходом. Он не успеет «почувствовать» последствия неправильного управления, что замедлит обучение и приведёт к нестабильному поведению. Использование «max_out_of_road_steps» создаёт более гибкую обучающую среду, в которой у агента есть шанс скорректировать свою траекторию, вернуться на полосу и получить положительное подкрепление за успешное исправление.

3.5 Роль предварительной обработки изображений против усложнения архитекуры нейронных сетей.

В ходе экспериментов с архитектурой нейронной сети для задачи удержания полосы движения было полезно протестировать, насколько простую модель можно использовать в связке с алгоритмом PPO, учитывая относительную простоту окружающей среды — отсутствие других автомобилей, препятствий или сложных перекрёстков. Изначально предполагалось, что для обработки изображений обязательно потребуется хотя бы небольшая свёрточная сеть, но, протестировав более простую модель, мы пришли к выводу, что она отлично справляется с поставленной задачей.

Итак, для данной задачи мы задаём архитектуру без свёрточных слоёв, описанную в пункте 2.4.3, где входом служит изображение с камеры в виде одномерного вектора. Такой подход позволяет проверить, способна ли базовая модель без глубокой обработки распознать достаточно информации из "сырых" данных и обучиться корректному поведению.

Результаты первых запусков показали, что обучение практически не происходило — модель не могла уловить закономерности в исходных изображениях и вырабатывала случайные действия, не демонстрируя прогресса даже после нескольких тысяч шагов. Это неудивительно, так как исходное изображение с камеры содержало множество лишних деталей и шумов, не несущих информации для задачи позиционирования по полосе.

Чтобы устранить этот барьер, была реализована простая функция предварительной обработки изображений, минимально, но эффективно подготавливающая данные для модели. Сначала изображение обрезается — остаётся только горизонтальная полоса в центральной части кадра, где расположена дорога (таким образом мы отбрасываем ненужные для данной задачи данные. Затем оно преобразуется в оттенки серого, сглаживается медианным фильтром, после чего применяется пороговая бинаризация по Отсу, которая превращает изображение в чёрно-белое.

Рассмотрим формально. Основная цель — максимально снизить размерность входных данных и выделить ключевую структурную информацию, минимизируя при этом шум.

Пусть исходное изображение — это трёхканальная матрица $I \in \mathbb{R}^{H \times W \times 3}$. Мы отбрасываем верхнюю и нижнюю части изображения, оставляя только строки с h_1 по h_2 включительно (3.5):

$$I_{\text{cropped}} = I[h_1 : h_2, :, :], \tag{3.5}$$

где h_1 = 32 — верхняя граница,
h_2 = 44 — нижняя граница.

Цветное изображение преобразуется в одноканальное изображение при помощи формулы яркости (3.6):

$$I_{\text{gray}}(x, y) = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B, \quad (3.6)$$

где R, G, B — значения соответствующих каналов в пикселе (x, y).

Для подавления шумов применяется медианный фильтр с ядром (3×3) . Для каждого пикселя вычисляется медиана значений в его окрестности (3.7):

$$I_{\text{median}}(x, y) = \text{median}\big(\{I_{\text{gray}}(x + i, y + j) \mid i, j \in \{-1,0,1\}\}\big) \quad (3.7)$$

Для получения чётких границ между дорожной разметкой и фоном используется пороговая бинаризация, при которой пиксели преобразуются к значениям 0 или 1 в зависимости от оптимального порога (T^*), найденного автоматически (3.8):

$$I_{\text{bin}}(x, y) = \begin{cases} 1, & \text{если } I_{\text{median}}(x, y) \geq T^*, \\ 0, & \text{иначе,} \end{cases} \quad (3.8)$$

где порог (T^*) определяется методом максимизации межклассовой дисперсии (3.9):

$$T^* = arg\max_T [\omega_1(T)\omega_2(T)(\mu_1(T) - \mu_2(T))^2], \quad (3.9)$$

где $\omega_1(T), \omega_2(T)$ — вероятности классов,
$\mu_1(T), \mu_2(T)$ — средние интенсивности классов.

Эта простая, но эффективная обработка позволила свести изображение к компактному и информативному вектору без необходимости использовать свёрточные слои. Благодаря этому подходу модель PPO обучилась в разы быстрее, чем при работе с необработанным изображением. Это подтверждает, что в простых средах с однозначной визуальной структурой предварительная фильтрация может полностью компенсировать отсутствие сложной архитектуры.

3.6 Результаты обучения и обобщающая способность модели

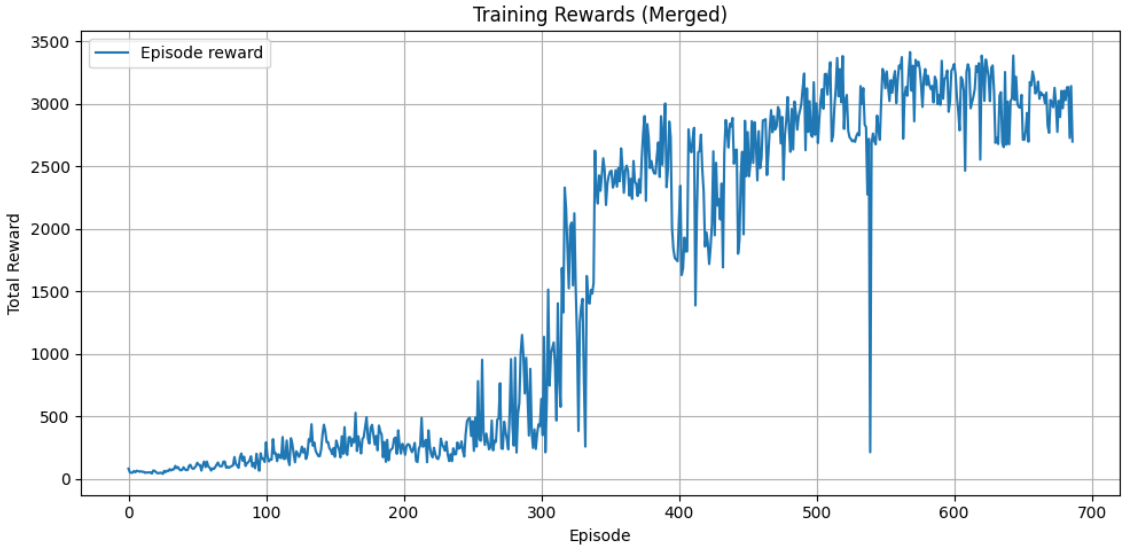

Рисунок 3.1 — График полной награды за эпизоды

В процессе обучения агента с использованием алгоритма PPO в симуляторе Webots велось логирование ключевых метрик, в частности, суммарной награды за каждый эпизод. Это позволило отслеживать прогресс обучения в динамике, а также выявить, насколько эффективно агент усваивает стратегию следования по полосе.

Обучение проходило в течение 527 эпизодов. В каждый эпизод агент стартовал с начальной позиции на обучающей трассе, состоящей из прямых участков и плавных поворотов. В процессе взаимодействия с окружением агент получал вознаграждение по правилам, описанным в пункте 3.2.

На рисунке 3.1 представлена динамика награды по эпизодам, которая демонстрирует характерный рост и стабилизацию обучения. Как видно, по мере увеличения количества эпизодов наблюдается рост накопленной награды, что свидетельствует о постепенном улучшении поведения агента. Мы видим, что график постоянно колеблется, это связано с тем, что агент пытается искать новые возможности и пробует новые варианты, но не забывает о предыдущих успехах. На 527 эпизоде мы видим резкую просадку вниз, это связано с тем, что для обучения стала использоваться новая сложная трасса, но при этом спустя несколько эпизодов график восстановился и находится на стабильно высоких значениях награды, несмотря на то, что всё ещё есть колебания из-за исследования моделью новых возможностей.

Вторая трасса отличалась более сложной геометрией: узкие повороты, чередующиеся прямые участки с криволинейными, изменение ширины дороги. Стоит отметить, что никаких изменений в архитектуре модели

произведено не было, модель использовала ту же систему вознаграждений и обработку наблюдений.

Результаты оказались впечатляющими: уже в первых 3–5 эпизодах на новой трассе модель продемонстрировала адаптивное поведение и успешное удержание автомобиля в пределах дорожного полотна. Это указывает на то, что модель усвоила не абсолютные координаты трассы, а общий принцип движения вдоль полосы, система наград и представление окружающей среды в виде бинаризованного изображения дороги обеспечили высокую переносимость знаний на новые условия для текущей задачи. Таким образом, можно сделать вывод, что даже при отсутствии сложных архитектур (таких как свёрточные слои), предварительная обработка изображения и хорошо спроектированная система вознаграждения позволяют достичь надёжного и обобщаемого поведения в простой задаче автономного управления. Кроме того, задачу легко обобщить на более сложные сценарии путём углубления используемых в алгоритме PPO нейронных сетей и прописывая желаемое поведение в функции наград. Это не потребует введения отдельных модулей, связанных рукописными интерфейсами, а может быть достигнуто сквозным обучением.

3.7 Ключевые выводы и пути дальнейшего развития

В главе 3 нами был разработан алгоритм, который позволят машине, самостоятельно выучить правила для движения на местности, а также улучшать себя при появлении новых условий и дорог несмотря на наличие пешеходов, плохих погодных условий, наличия потока транспорта. Начальная версия этого алгоритма была построена и протестирована в среде виртуального моделирования Webots. Предложенный алгоритм легко обобщается на новые условия путём усложнения архитектуры нейросетей и модификации функции наград.

ЗАКЛЮЧЕНИЕ

В данной работе была рассмотрена задача оптимизации алгоритмов распознавания дорожной разметки и управления движением автомобиля в сложных погодных условиях с использованием классических алгоритмов компьютерного зрения, глубокого обучения и обучения с подкреплением.

В первой главе был реализован прототип системы удержания полосы на основе базовых алгоритмов компьютерного зрения (фильтрация изображений, эрозия, дилатация, преобразование Хафа) и протестирован в симуляционной среде Webots. Алгоритм продемонстрировал способность удерживать полосу движения и объезжать препятствия, однако оказался ограничен в универсальности и устойчивости при сложных погодных условиях. Оценка производительности системы с использованием метрик LDE и CPA позволила количественно подтвердить её эффективность в контролируемой среде.

Во второй главе проведён обзор современных методов машинного обучения и их применимости к задаче автопилотирования. Рассмотрены преимущества и недостатки различных подходов: модульной архитектуры, сквозного обучения и гибридных систем, а также затронута проблема сбора и генерации обучающих данных, особенно для редких и опасных ситуаций. Особое внимание уделено симуляционному обучению, как способу масштабируемого и безопасного получения ценных данных для тренировки моделей.

В третьей главе разработана и обучена система управления на основе методов обучения с подкреплением, что позволило избавиться от необходимости в размеченных данных и повысить гибкость системы. Была обучена модель, способная эффективно управлять автомобилем, адаптируясь к изменениям окружающей среды и обучаясь взаимодействовать с ней в процессе симуляции. Система продемонстрировала высокую устойчивость к различным условиям, включая отсутствие разметки и плохую видимость, а также мы показали, как её можно адаптировать под более сложные задачи.

Таким образом полученные в работе результаты могут быть использованы в дальнейших исследованиях по созданию более интеллектуальных и адаптивных систем автономного управления транспортными средствами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Aghajanyan, A., Gündüz, D. Reinforcement Learning for Diplomacy: A New Approach to Strategic Decision Making / A. Aghajanyan, D. Gündüz // arXiv.org. – 2022. – Mode of access: https://arxiv.org/abs/2210.05492. – Date of access: 20.04.2025.

2. Caltagirone, L., Bellone, M., Svensson, L., Wahde, M. LiDAR-based Driving Path Generation Using Fully Convolutional Neural Networks [Electronic resource] L. Caltagirone, M. Bellone, L. Svensson, M. Wahde // IEEE Intelligent Vehicles Symposium (IV). – 2017. – Mode of access: https://ieeexplore.ieee.org/document/7995823. – Date of access: 20.04.2025.

3. Cybenko, G. V. Approximation by Superpositions of a Sigmoidal Function / G. V. Cybenko // Mathematics of Control, Signals, and Systems. – 1989. – Vol. 2, No. 4. – P. 303–314.

4. Duda, R. O., Hart, P. E. Use of the Hough Transformation to Detect Lines and Curves in Pictures / R. O. Duda, P. E. Hart // Artificial Intelligence Center. – 1971. – (Technical Note 36); published in: Communications of the ACM. – 1972. – Vol. 15, No. 1. – P. 11–15 [Electronic resource]. – Mode of access: http://www.ai.sri.com/pubs/files/tn036-duda71.pdf. – Date of access: 10.04.2025.

5. Ghosh, S., Das, N. Autoencoder Based Unsupervised Feature Selection for High-Dimensional Data / S. Ghosh, N. Das // IEEE Transactions on Neural Networks and Learning Systems. – 2022. – Vol. 33, No. 4. – P. 1398–1411.

6. Hu, R., Yang, L., Zeng, W. Motion Inspired Unsupervised Perception and Prediction in Autonomous Driving / R. Hu, L. Yang, W. Zeng // Transportation Research Part C: Emerging Technologies. – 2021. – Vol. 132. – P. 103361[Electronic resource]. – Mode of access: https://www.sciencedirect.com/science/article/abs/pii/S0968090X21004769. – Date of access: 20.04.2025.

7. Kalra, N., Paddock, S. M. Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability? // RAND Corporation [Electronic resource]. – 2016. – Mode of access: https://www.rand.org/pubs/research_reports/RR1478.html. – Date of access: 10.04.2025.

8. Lan, G., Hao, Q. End-To-End Planning of Autonomous Driving in Industry and Academia: 2022–2023 [Electronic resource] / G. Lan, Q. Hao // arXiv.org. – 2024. – Mode of access: https://arxiv.org/abs/2401.08658. – Date of access: 20.04.2025.

9. Markov Decision Process and Bellman Equations [Electronic resource] / Stanford University // Stanford University. – 2021. – Mode of access: https://web.stanford.edu/class/cs234/modules.html. – Date of access: 21.04.2025.

10. Road Traffic Deaths: Mortality Rate (per 100,000 population) [Electronic resource] / World Health Organization. – Geneva, 2024. – Mode of access: https://data.who.int/ru/indicators/i/B9D9E6A/D6176E2. – Date of access: 09.04.2025.

11. SAE J3016: Levels of Driving Automation / SAE International. Committee on On-Road Automated Driving (ORAD). – 2021. – 41 p. – DOI: 10.4271/J3016_202104.

12. Sampaio, R. A., Garcia, J. D., Poggi, M., Vidal, T. Regularization and Optimization in Model-Based Clustering [Electronic resource] / R. A. Sampaio, J. D. Garcia, M. Poggi, T. Vidal // arXiv.org. – 2023. – Mode of access: https://arxiv.org/abs/2302.02450. – Date of access: 20.04.2025.

13. Sharma, A. Understanding Activation Functions in Neural Networks [Electronic resource] / A. Sharma // The Theory Of Everything. – 2017. – Mode of access: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0. – Date of access: 15.04.2025.

14. Silver, D. Lecture 2: Markov Decision Processes [Electronic resource] / D. Silver // University of Alberta. – 2015. – Mode of access: https://www.youtube.com/watch?v=lfHX2hHRMVQ. – Date of access: 21.04.2025.

15. Sutton, R. S., Barto, A. G. Reinforcement Learning: An Introduction / R. S. Sutton, A. G. Barto. – 2nd ed. – Cambridge, MA: MIT Press, 2018. – Chapter 10, pp. 243–256; Chapter 11, pp. 257–284.

16. Tesla AI Day 2021[Electronic resource] // Tesla, Inc. – 2021. – Mode of access: https://www.youtube.com/watch?v=j0z4FweCy4M. – Date of access: 20.04.2025.

17. Tesla AI Day 2022[Electronic resource] // Tesla, Inc. – 2022. – Mode of access: https://www.youtube.com/watch?v=ODSJsviD_SU. – Date of access: 20.04.2025.

18. Wang, Q., Yu, L. RAISocketAI: Reinforcement Learning for RTS Game Micro-Management / Q. Wang, L. Yu // arXiv.org. – 2024. – Mode of access: https://arxiv.org/abs/2402.08112. – Date of access: 20.04.2025.

19. Yoon, J., Kim, T. Human-Robot Collaboration Using Reinforcement Learning in Real-World Environments / J. Yoon, T. Kim // Live Science. – 2025. – Mode of access: https://www.livescience.com/technology/robotics/watch-eerie-video-of-army-of-humanoid-robots-marching-naturally-thanks-to-a-major-ai-upgrade. – Date of access: 20.04.2025.

20. Zhang, S., Liu, Y. Overcooked: An Interactive Reinforcement Learning Approach to Multi-agent Collaboration / S. Zhang, Y. Liu // arXiv.org. – 2023. – Mode of access: https://arxiv.org/abs/2305.16708. – Date of access: 20.04.2025.

21. Zhang, Y., Chen, X. Anomaly Detection in High-Dimensional Data Using Isolation Forest and One-Class SVM / Y. Zhang, X. Chen // Journal of Machine Learning Research. – 2022. – Vol. 23, No. 1. – P. 1–25.

22. Zhou, Y., Gallivan, K. A., Barbu, A. Scalable Clustering: Large Scale Unsupervised Learning of Gaussian Mixture Models with Outliers [Electronic resource] / Y. Zhou, K. A. Gallivan, A. Barbu // arXiv.org. – 2023. – Mode of access: https://arxiv.org/abs/2302.14599. – Date of access: 20.04.2025.