	<p>Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)</p>
---	--

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Теоретическая информатика и компьютерные технологии

## **ОТЧЁТ ПО ПРЕДДИПЛОМНОЙ ПРАКТИКЕ НА ТЕМУ:**

Исследование осцилляторных нейронных сетей  
для анализа больших данных

Студент ИУ9-82Б

\_\_\_\_\_

*подпись, дата*

Жук Д.О.

\_\_\_\_\_

*фамилия, и.о.*

Научный руководитель

\_\_\_\_\_

*подпись, дата*

Каганов Ю.Т., к.т.н.

\_\_\_\_\_

*фамилия, и.о.*

2023 г.

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	3
1. ТЕОРЕТИЧЕСКАЯ БАЗА .....	6
1.1. Персептрон.....	6
1.2. Осцилляторная нейронная сеть .....	8
1.3. Обучение нейронной сети .....	12
2. РАЗРАБОТКА .....	16
3. РЕАЛИЗАЦИЯ .....	19
3.1. Осцилляторная нейронная сеть .....	20
3.2. Движение автомобиля .....	22
3.3. Отрисовка сцены .....	26
3.4. Обработка пользовательского ввода .....	27
3.5. Создание трассы .....	27
3.6. Обработка коллизий.....	28
3.8. Получение данных для нейронной сети .....	30
3.9. Создание обучающей выборки .....	34
3.10. Логика пользовательского интерфейса.....	35
4. ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ .....	42
ЗАКЛЮЧЕНИЕ .....	46
СПИСОК ЛИТЕРАТУРЫ.....	49
ПРИЛОЖЕНИЕ А. Реализация ОНС .....	51
ПРИЛОЖЕНИЕ Б. Реализация автомобиля .....	52
ПРИЛОЖЕНИЕ В. Отрисовка в пространстве полигона и автомобиля .....	54
ПРИЛОЖЕНИЕ Г. Обработка пользовательского ввода.....	56

## ВВЕДЕНИЕ

Нейронная сеть – это один из ключевых элементов в развитии современных технологий и общества в целом. Нейронные сети используются в широком спектре задач – от распознавания образов до принятия решений в медицине и экономике. Взаимодействие человека с технологиями на основе нейронных сетей становится все более распространенным и влияет на общество в целом. Большинство людей в нашей цифровой эре уже сталкивались с нейронными сетями, даже не подозревая об этом. Например, системы рекомендаций, которые можно наблюдать на различных интернет-площадках, применяют нейронные сети для предсказания предпочтений пользователей.

Использование нейронных сетей не ограничивается только лишь развлекательной областью. Уже сейчас они внедрены во множество сфер: врачами для постановки диагноза, инженерами для распределения нагрузки в энергосети, финансовыми аналитиками для предсказания поведения рынка. Нейронные сети активно применяются в машинном переводе, что позволяет существенно улучшить качество перевода текстов на различные языки.

Высокая стоимость обслуживания ограничивает доступ к нейронным сетям для малых и средних предприятий. Некоторые компании могут использовать готовые модели нейронных сетей, чтобы избежать больших вложений в их создание и обучение.

Рассматривая проблему дороговизны нейронных сетей, становится очевидно, что одной из основных статей расходов являются затраты на электроэнергию. Самые передовые искусственные нейронные сети не только отстают по размерам на пару порядков от биологических, но и требуют, как минимум на три порядка больше электроэнергии для своей работы [1].

Потребность в аппаратных ресурсах для обучения нейронных сетей растет гораздо быстрее, чем эти аппаратные ресурсы появляются (рисунок 1). А стоимость одной операции обучения известной языковой модели GPT-3 начинается от четырех с половиной миллионов долларов США [2].

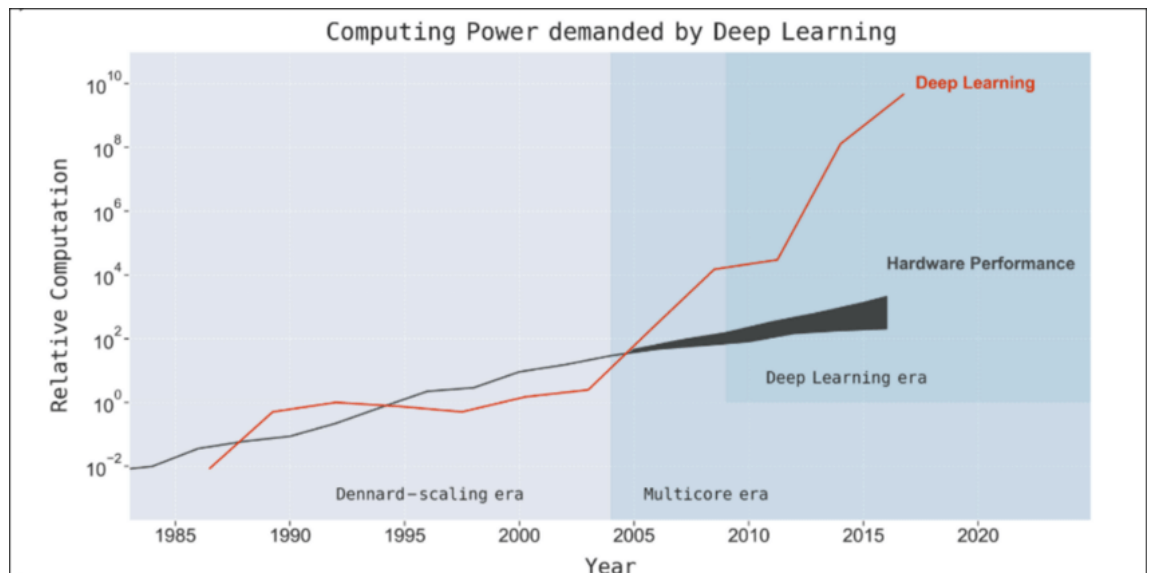


Рисунок 1 – График зависимости сложности и возможности современных вычислительных ресурсов от времени

Возникает основная проблема: классические искусственные нейронные сети содержат на фундаментальном уровне проблемы и ограничения, которые в связи с многократным масштабированием системы, начинают играть весомую негативную роль. Некоторые из этих упущений могут быть разрешены при переходе от классических нейронных сетей к осцилляторным.

Осцилляторные нейронные сети (ОНС) представляют собой новый тип искусственных нейронных сетей, который отличается от традиционных методов машинного обучения. В отличие от классических методов, таких как регрессия и статистические модели, которые используют статические функции для аппроксимации данных, ОНС используют динамические процессы для анализа временных рядов. В отличие же от классических нейронных сетей осцилляторная нейронная сеть обладает высокой устойчивостью к шуму и может работать с шумными данными, не теряя точность и может обучаться быстрее, чем классические персептроны, благодаря использованию колебательных свойств.

В данной работе рассмотрена теория классических нейронных сетей в целом и осцилляторных в частности, объяснена причина создания и реализована

ОНС. Разработана программная реализация алгоритма построения и тестирования на основе сгенерированных и реальных данных.

Целью работы является создание приложения, реализующего логику работы осцилляторной нейронной сети для анализа больших данных.

# 1. ТЕОРЕТИЧЕСКАЯ БАЗА

## 1.1. Персептрон

Ученых всего мира многие годы интересовал вопрос возможности повторения логики работы мозга и создание искусственного интеллекта. Одним из первых важным шагов в этом направлении стало создание модели искусственного нейрона.

Модель искусственного нейрона чаще всего фигурирует как блок обработки информации нейронной сети. Искусственные нейроны действительно примитивны по сравнению с известными данными о реальных нейронах в мозгу. Модель нейрона обычно отражает три важные характеристики реальных нейронов:

1. Взвешенные значения синапсов;
2. Использование нелинейной функции активации;
3. Ограничение допустимой амплитуды выхода нейрона.

Модель искусственного нейрона, изображенная на рисунке 2, была предложена Маккаллохом и Питтсом в 1943 г.

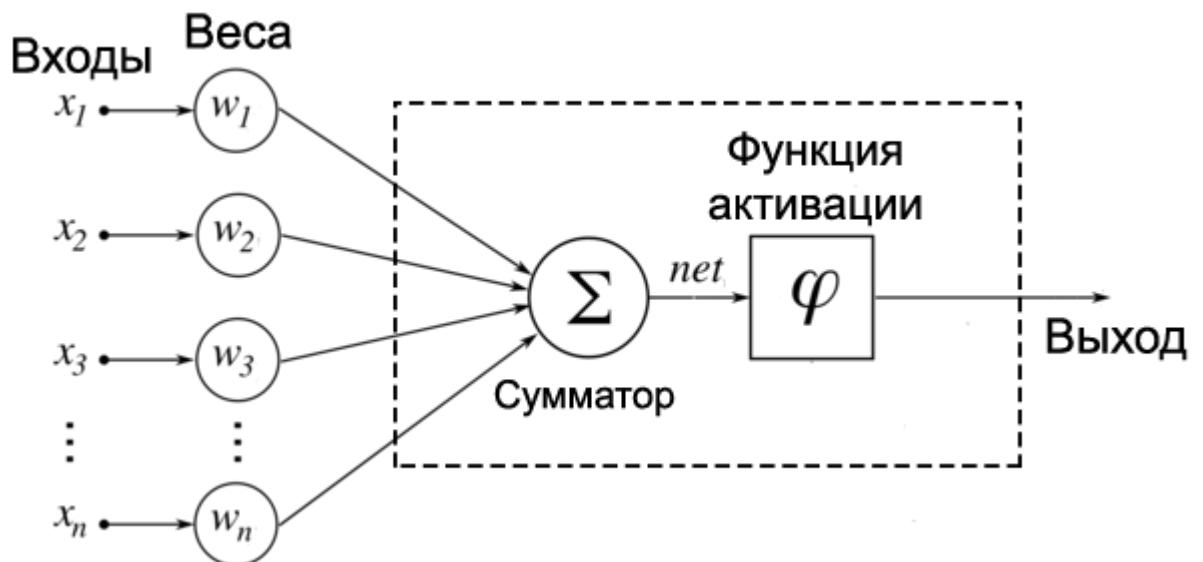


Рисунок 2 – Формальный нейрон Маккаллоха-Питтса

Для нейрона с  $n$  входами  $x_1, \dots, x_n$  и входными синапсами с весами  $w_1, \dots, w_n$ , выход нейрона  $y$  определяется как

$$y(t+1) = G\left(\sum_{j=1}^n \omega_j x_j(t) - h_j\right) \quad (1)$$

где  $t$  – дискретное время,  $G(x)$  – функция активации нейрона, а  $h_j$  – порог [3].

Если используются  $x_j \in \{0, 1\}$ , то самым простым  $G(x)$  является ступенчатая функция Хэвисайда, тогда как при  $x_j \in \{-1, 1\}$  естественным является использование функции активации  $G(x) = \text{sign } x$ . Также используются непрерывные сигмовидные функции. Простой пример сигмоиды функции активации — это логистическая функция:

$$G(x) = \frac{1}{1 + e^{-\alpha x}} \quad (2)$$

На рисунке 2 изображена однослойная нейронная сеть из  $n$  одинаковых формальных нейронов, не связанных между собой, каждый из которых имеет  $n$  входов и один выход. Пусть  $W_{jk}$  — вес соединения входа  $j$  с нейроном  $k$  и  $h_1, \dots, h_n$  — пороги нейронов. Тогда выходы  $y_1, \dots, y_m$  сети прямого распространения (также называемой однослойным персептроном) определяются как:

$$y_i = G\left(\sum_{k=1}^n W_{ik} x_k - h_i\right), \quad i = 1, \dots, m, \quad (3)$$

или в векторном виде

$$y = G(\hat{W}x - h) \equiv \hat{T}x, \quad (4)$$

где  $x = (x_1, \dots, x_n)^T$ ,  $y = (y_1, \dots, y_m)^T$ ,  $h = (h_1, \dots, h_n)^T$  и  $\hat{W} = [W_{jk}]$  является матрицей  $m \times n$ .

Пусть ряд однослойных персептронов соединен в каскад – цепочку последовательно соединенных однослойных сетей, так что выход предыдущей сети служит входом последующей сети. Пусть  $n^{(l)}$  будет количеством нейронов в слое  $l$ . Тогда мы имеем  $\dim x^{(l)} = n^{(l)}$  и  $\hat{W}^{(l)}$  – это  $n^{(l)} \times n^{(l-1)}$ -матрица. Пример многослойного персептрона (каскада нескольких однослойных сетей прямого распространения) изображен на рисунке 3.

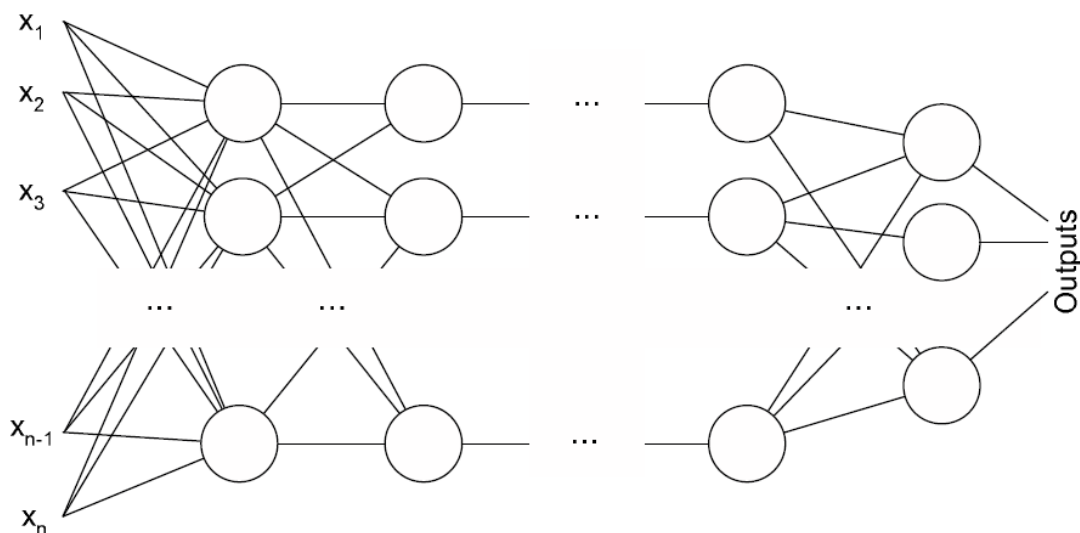


Рисунок 3 – Архитектура многослойного перцептрона

В настоящее время персептроны широко используются для разработки различных параллельных вычислительных алгоритмов, которые оказываются более эффективными, чем классические сеточные алгоритмы или методы конечных элементов [4].

## 1.2. Осцилляторная нейронная сеть

Мозг представляет собой высокораспределённую систему, в которой параллельно выполняются многочисленные операции. Один из важных вопросов заключается в том, как координируются вычисления, происходящие одновременно в пространственно разделённых областях обработки. Динамическое связывание через синхронизацию активности нейронов может играть роль координирующего механизма.

Синхронизация была экспериментально обнаружена в различных структурах мозга. Экспериментальное обнаружение синхронных колебаний усилило внимание к колебательным аспектам обработки зрительной информации. Как выяснилось, синхронизация, сопровождающая работу зрительной коры при обработке изображений, обладает следующими особенностями:

1. Синхронизация является стимулозависимым явлением;



2. Синхронизация достигается за счет нейронных взаимодействий;
3. Дальние связи нейронов ответственны за связывание через синхронизацию [5].

Осцилляторные нейронные сети – это класс искусственных нейронных сетей, которые способны генерировать периодические сигналы (осцилляции) в своих выходах. Они используются в различных областях, таких как управление роботами, обработка потока изображений, распознавание речи и других.

Осцилляторные нейронные сети имеют ряд преимуществ перед другими типами нейронных сетей, например возможность генерации периодических сигналов и высокая степень гибкости в управлении динамическими системами. ОНС могут использоваться для моделирования биологических процессов, таких как сердцебиение, дыхание, генерации звуков и музыки, анализа временных рядов и многих других задач.

Основой осцилляторной нейронной сети являются нейроны, которые могут генерировать периодические сигналы с помощью обратной связи. Эти нейроны могут быть реализованы с использованием различных типов моделей, включая нейроны ФитцХью-Нагумо, волновые уравнения и другие.

Одна из наиболее распространенных форм осцилляторной нейронной сети – это нейронные сети Кохонена, которые имеют два типа нейронов: экзиторные (стимулирующие) и ингибиторные (тормозящие). Эти нейроны соединены с помощью обратных связей, которые позволяют им генерировать периодические сигналы.

Поведение осциллятора определяется следующим уравнением:

$$\frac{d\varphi_i}{dt}(t) = \omega_i + \sum_{j=1}^N K_{ij}H_{ij}[\varphi_j(t) - \varphi_i(t)] \quad (5)$$

где  $\varphi_i$  — фаза осциллятора  $i$ ,  $\omega_i$  — его собственная частота,  $K_{ij}$  — вес связи,  $H_{ij}$  — функция связи между осциллятором  $i$  и  $j$ .

В общем, выбирая соответствующую функцию связи  $H_{ij}$ , любое произвольно сложное поведение может обеспечить стабильное состояние

кластера. В частности, когда  $H_{ij}[\varphi_j(t) - \varphi_i(t)] = \sin[\varphi_j(t) - \varphi_i(t)]$ , тогда система становится моделью Курамото. Динамика слабо связанных осцилляторов может демонстрировать сложное хаотическое поведение [6].

Получается, что определяющим отличием от персептрона является взаимодействие нейронов (осцилляторов) в пределах одного слоя, а также отсутствие дискретного момента вычислений (рисунок 3).

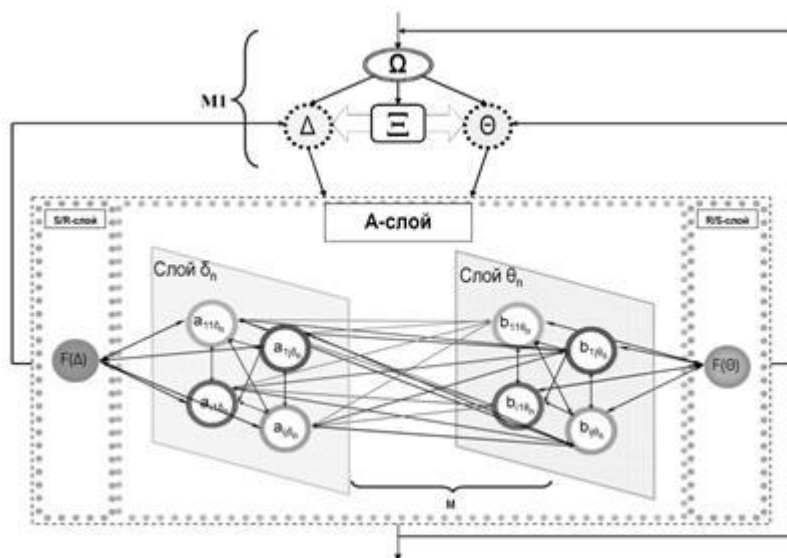


Рисунок 3 – Обобщённая модель осцилляторной нейронной сети  
гомеостатического типа

Дополнительно, осцилляторные нейронные сети могут быть использованы для решения задач, связанных с управлением роботами, например, для создания более точных и быстрых движений, повышения эффективности и безопасности производства. Кроме того, ОНС могут использоваться для обработки потоков изображений и видео, что может быть полезно для автоматического распознавания объектов и сцен, а также для улучшения их качества.

Ещё одной областью применения осцилляторных нейронных сетей является распознавание речи. Благодаря возможности генерации периодических сигналов ОНС могут использоваться для синтеза речи, а также для улучшения качества распознавания и обработки речевых сигналов.

Одним из преимуществ осцилляторных нейронных сетей является их способность моделировать сложные динамические системы с высокой степенью

точности. Это позволяет использовать ОНС для анализа временных рядов, прогнозирования и управления процессами в различных областях, таких как финансы, метеорология и т.д. [7].

Для более глубокого понимания осцилляторных нейронных сетей, необходимо рассмотреть их структуру и принципы работы. ОНС состоят из нейронов, которые соединены между собой синапсами. Каждый нейрон в ОНС имеет свой внутренний осциллятор, который генерирует периодический сигнал. Эти сигналы передаются по синапсам от одного нейрона к другому, образуя циклические паттерны активности.

Одной из ключевых особенностей ОНС является их способность к самоорганизации и адаптации. В процессе обучения ОНС могут изменять свою структуру и параметры, чтобы адаптироваться к изменяющимся условиям и задачам. Это делает ОНС более гибкими и эффективными для решения различных задач.

Определяющими факторами осцилляторной нейронной сети связанных с набором ее свойств и характеристик, является:

1. Осцилляторность. Основным свойством ОНС является способность к самоподдерживающейся осцилляции, т.е. периодическому колебанию активности нейронов внутри сети без внешнего воздействия. Осцилляторность может быть реализована различными механизмами, включая положительную обратную связь, ингибирование и диффузионную динамику [7].

2. Глобальная синхронизация. ОНС обладает способностью к глобальной синхронизации, т.е. координированному колебанию активности всех нейронов внутри сети. Это свойство особенно важно для задач, связанных с управлением и координацией.

3. Гибкость. ОНС может иметь гибкую структуру и связи, которые могут изменяться во времени и в ответ на внешние воздействия. Это позволяет ОНС адаптироваться к изменяющейся среде и эффективно решать различные задачи.

4. Параллелизм. ОНС может обрабатывать информацию параллельно, что позволяет ей эффективно решать задачи с высокой степенью параллелизма, такие как распознавание образов, управление манипуляторами и т.д.

5. Робастность. ОНС может обеспечивать высокую робастность к помехам и внешним воздействиям благодаря особенностям структуры и динамики. Это позволяет ей эффективно работать в условиях переменной и ненадежной среды.

6. Низкое энергопотребление. ОНС имеет низкое энергопотребление, благодаря своей способности к эффективной обработке информации и использованию энергии только в тех нейронах, которые активны в данный момент [8].

### 1.3. Обучение нейронной сети

Обучение нейронной сети – это процесс, в котором сеть настраивается на определенный набор данных. Она извлекает закономерности из этих данных и использует их для решения задач.

Для обучения нейронной сети необходимо выполнить несколько этапов. Во-первых, необходимо подготовить данные для обучения. Это может включать в себя очистку данных, преобразование формата и нормализацию.

Во-вторых, данные делятся на три группы: данные для обучения, данные для проверки и данные для тестирования. Данные для обучения используются для настройки весов нейронной сети. Данные для проверки – для определения эффективности обучения, а данные для тестирования используются для оценки общей производительности нейронной сети.

Далее происходит сам процесс обучения, который может быть выполнен различными способами. Например, нейронные сети могут быть обучены с помощью алгоритма обратного распространения ошибки, генетических алгоритмов или методов оптимизации, таких как градиентный спуск.

Алгоритм обратного распространения ошибки – это один из наиболее распространенных методов обучения нейронных сетей. Он основан на

минимизации функции потерь, которая определяет, насколько хорошо нейронная сеть выполняет задачу.

Процесс обучения включает в себя следующие шаги:

1. Инициализация весов нейронной сети случайными значениями;
2. Подача обучающих данных на вход сети и вычисление выходных значений;
3. Вычисление ошибки сети с помощью функции потерь;
4. Распространение ошибки обратно через сеть, чтобы вычислить градиент функции потерь по весам;
5. Обновление весов сети с помощью градиентного спуска, для минимизации функции потерь;
6. Повторение шагов 2–5 для всех обучающих данных в течение нескольких эпох.

Алгоритм обратного распространения ошибки имеет ряд преимуществ:

- Может использоваться для обучения различных типов нейронных сетей, включая многослойные перцептроны, рекуррентные нейронные сети и свёрточные нейронные сети;
- Может быть применен к различным типам задач, включая классификацию, регрессию, сегментацию и обработку естественного языка;
- Может быть эффективно реализован на современных графических процессорах, что позволит ускорить обучение.

Однако у алгоритма обратного распространения ошибки есть и некоторые недостатки:

- Может привести к переобучению, когда модель слишком хорошо подстраивается под обучающие данные и не может приспособиться к новым данным;
- Может застрять в локальных минимумах функции потерь, что может привести к недостаточно хорошим результатам;
- Может быть медленным в обучении больших нейронных сетей, особенно если используется стандартный градиентный спуск.

Q-обучение – это метод обучения с подкреплением, который позволяет агенту находить оптимальную стратегию поведения в среде. Он основан на оценке функции ценности действий агента, которая называется функцией Q.

Функция Q определяет ожидаемую награду, которую агент получит, если он выберет определенное действие в данном состоянии. Она вычисляется как сумма награды, полученной агентом за выполнение действия, и ожидаемой награды, которую агент получит, если он продолжит действовать оптимальным образом.

Q-обучение осуществляется путем оценки функции Q методом временных разностей. Этот метод заключается в том, что агент использует текущую оценку функции Q для обновления ее значения на основе полученной награды и следующего состояния. Таким образом, функция Q постепенно уточняется в процессе обучения.

Нейроэволюция – это метод обучения нейронных сетей, который использует эволюционные алгоритмы для создания и оптимизации архитектуры сети. В отличие от традиционных методов обучения, которые требуют ручной настройки архитектуры сети, нейроэволюция позволяет автоматически создавать оптимальную архитектуру сети.

Процесс нейроэволюции состоит из нескольких этапов. Сначала создается начальная популяция нейронных сетей, которая может быть создана случайным образом или с использованием заранее определенных правил. Затем каждая нейронная сеть в популяции оценивается на основе ее производительности в решении задачи. Это может быть выполнено путем оценки точности предсказания сети на наборе данных для обучения или путем оценки ее производительности в среде с подкреплением.

Далее происходит процесс эволюции, во время которого выбираются наиболее успешные сети и используются для создания новой популяции. Этот процесс может включать в себя различные операции, такие как скрещивание, мутация и отбор, которые позволяют создавать новые и более успешные нейронные сети.

В процессе обучения нейронная сеть постепенно настраивает свои веса, чтобы минимизировать ошибку и повысить точность предсказаний. После достижения необходимой точности на данных для обучения нейронная сеть проверяется на данных для проверки и тестирования, чтобы убедиться, что она не переобучилась и способна обобщать данные.

## 2. РАЗРАБОТКА

Для решения задачи исследования осцилляторных нейронных сетей необходимо сделать следующие шаги:

1. Реализовать осцилляторную нейронную сеть и определить корректность работы;
2. Разработать задачу и способ получения данных пригодных для решения как классической, так и осцилляторной нейронной сетью;
3. Выбрать метод и обучить на нем обе нейронных сети в соответствии с поставленной задачей.

Рассмотрим каждый этап подробно.

1. Реализовать осцилляторную нейронную сеть и определить корректность работы.

Рассмотренные популярные библиотеки, реализующие работу с нейронными сетями такие как Brain.js, TesnserFlow.js, Neuro.js, не предоставляют функционал для работы с осцилляторными нейронными сетями. Это приводит к тому, что осцилляторную нейронную сеть предстоит реализовывать самостоятельно.

Интересной задачей является проверка самого получившегося кода на факт того является ли получившаяся модель действительно осцилляторной нейронной сетью, ведь отсутствует эталонная реализация для сравнения. В связи с этим было решено проводить проверку на основе найденных экспериментов проведенных ранее другими исследователями.

2. Разработать задачу и способ получения данных пригодных для решения как классической, так и осцилляторной нейронной сетью.

После того как получили корректно работающую ОНС, необходимо выбрать задачу посильно решить которую обеим нейронными сетям. Такой была выбрана проблема прохождения трассы автомобилем. За критерии качественного прохождения трассы по приоритетно будут являться следующие факторы:



1. Отсутствие столкновений машины с трассой;
2. Прохождение маршрута;
3. Эффективность и время работы разработанного пользовательского приложения;
4. Скорость прохождения пути.

У решения проблемы в данной постановке множество плюсов. Это позволяет иметь и создавать большое количество данных, которые в то же время легко анализируются человеком. Также это позволяет реализовать наглядную модель, демонстрирующую качество получившихся результатов.

3. Выбрать метод и обучить на нем обе нейронных сети в соответствии с поставленной задачей.

В итоге имея модель гонки требуется обучить на ней ездить обе нейронные сети. Для этого потребуется рассмотреть несколько методов тестирования нейронных сетей, их достоинства и недостатки и выбрать финальную реализацию. Было рассмотрено три типа тестирования: обучение на основе обучающей выборки, Q-обучение и нейроэволюция.

Для тренировки на обучающей выборке важным плюсом является то, что популярные библиотеки реализуют такой вид обучения. Однако данные предстоит сгенерировать самостоятельно. Это скажется на факте скорости который хоть и менее приоритетный, но является фактором качества.

У Q-обучения за плюс можно считать возможность в случае его реализации будет получено оптимальное по скорости прохождения трассы решения. Это достигается за счет того, что нейросеть будет стараться улучшать свои результаты. В то же время это крайне сложный для реализации способ обучения, которые популярные библиотеки не реализуют в силу необходимости слишком тесной связи с моделью.

Нейроэволюция из рассмотренных методов является самой простой в оценке качества обучающейся нейронной сети – выживает и дает потомство сильнейшая. Однако в этом же заключается и главная слабость – она на порядки медленнее других [9].

В связи с чем было принято решение разработать обучение на обучающей выборке.

### 3. РЕАЛИЗАЦИЯ

Алгоритмы, рассмотренные в работе, были применены на практике с помощью следующих инструментов:

1. Язык программирования TypeScript;
2. Brain.js – библиотека JavaScript, используемая для обучения нейронных сетей;
3. Function Plot – библиотека JavaScript, используемая для построения графиков;
4. Стандартизированный язык разметки документов HTML5;
5. Язык таблицы стилей CSS для описания внешнего вида HTML – страницы.

Рассмотрим библиотеку Brain.js подробнее. Пакет Brain.js используется для создания и обучения нейронных сетей. Этот пакет позволяет свободно манипулировать с обучением нейронной сети, сохранять и выгружать обученную модель в файл [10].

Рассмотрим библиотеку Function Plot подробнее. Function Plot — это библиотека построения графиков, созданная на основе D3.js и используемая для визуализации функций с небольшой настройкой.

В настоящее время библиотека поддерживает интерактивные линейные диаграммы и диаграммы рассеяния. Каждый раз, когда масштаб графика изменяется, функция вычисляется снова с новыми границами. Это приводит к возможности строить бесконечные графики.

Function Plot в отличие от других библиотек, использующих  $n$ -равноотстоящие друг от друга точки, соединенные отрезками, Function Plot использует интервальную арифметику для правильного определения участков экрана, которые необходимо отобразить с помощью нескольких выборок [11].

Была реализована собственная надстройка над компилятором TypeScript производящая необходимый пласт работ над получившимися JavaScript файлами для корректной работы в браузерной среде. Например, комментирование

глобальных импортов и переименование путей импортов, чтобы они содержали расширение файла. Разработчики TypeScript считают, что это не в их зоне ответственности и позиционируют себя отдельно от корректной сборки под соответствующие окружения JavaScript [12].

### 3.1. Осцилляторная нейронная сеть

Исходя из поставленной задачи практической работы требуется реализовать осцилляторную нейронную сеть на основе изученной теории и воплотить способ проверки корректности её работы.

Если первый пункт не вызывает излишних сложностей в силу простоты формулы (5) при выборе модели Курамото, то второй – уже не является столь тривиальным. Однако выход был найден – было решено повторить эксперимент, предложенный в статье исследующей осцилляторные нейронные сети [13].

Для этого была незначительно модифицирована получившаяся реализация ОНС, запущена для заданных данных и проверена на совпадение получившихся графиков (приложение А). Рассмотрим каждый этап подробнее.

В ОНС формула (5) была заменена на формулу

$$\frac{d\varphi_i}{dt}(t) = 2\pi \left( f_i + \sum_{j=1}^N \varepsilon_i \varepsilon_j \sin[\varphi_j(t) - \varphi_i(t)] \right) \quad (6)$$

Формула (6) отличается от формулы (5), однако существует прямая замена одних переменных на другие, что подтверждает их эквивалентность:

$$\omega_i = 2\pi f_i, \quad K_{ij} = 2\pi \varepsilon_i \varepsilon_j \quad (7)$$

В статье [13] предлагается использовать данные значения  $f$  и  $\varepsilon$ , чтобы построить график зависимости частоты от времени:

$$f = [21 \quad 14 \quad 10 \quad 15 \quad 20 \quad 25 \quad 12.5 \quad 17.5 \quad 17.5 \quad 22.5],$$

$$\varepsilon = [4 \quad 4 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1].$$

После чего получаем график, изображенный на рисунке 4.

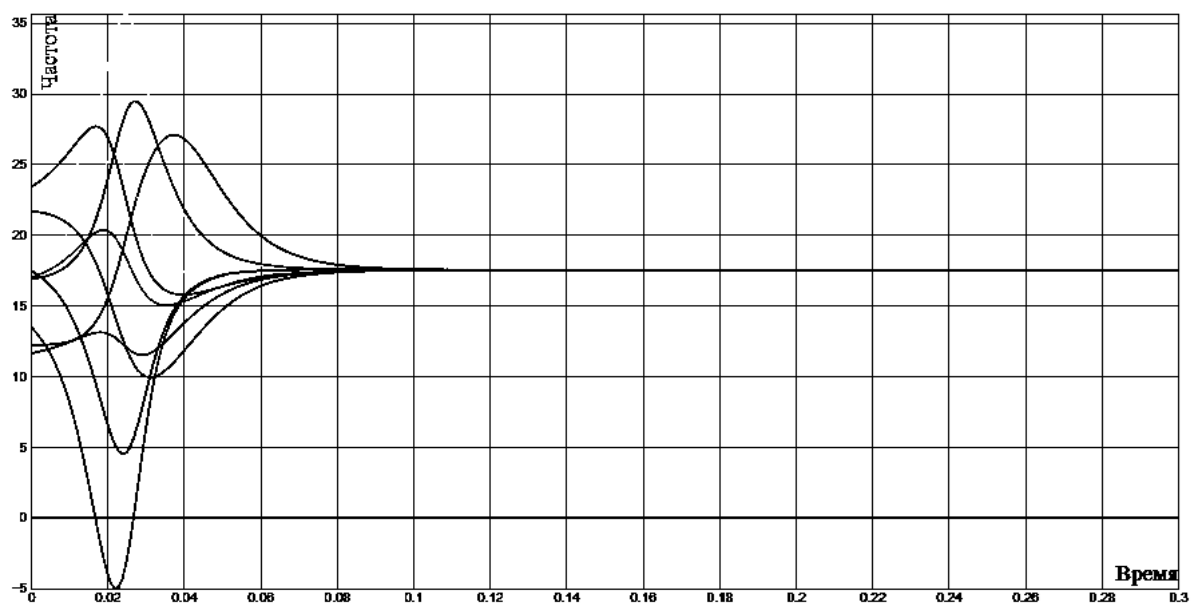


Рисунок 4 – график получившейся осцилляторной нейронной сети при заданных данных

Сравниваем его с графиком из статьи (рисунок 5) – получаем, что данные корректны.

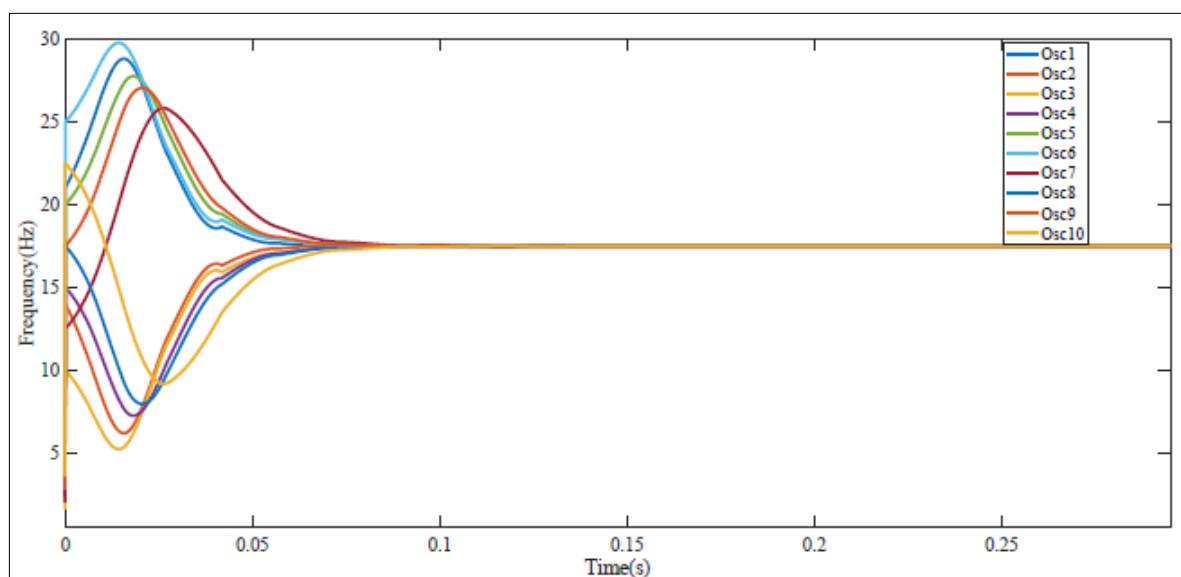


Рисунок 5 – график из статьи [13]

Сравним второй график, который получается под действием изменения девятого значения в массиве  $f$  с 17.5 на 40 – он тоже является корректным (рисунки 6 и 7).

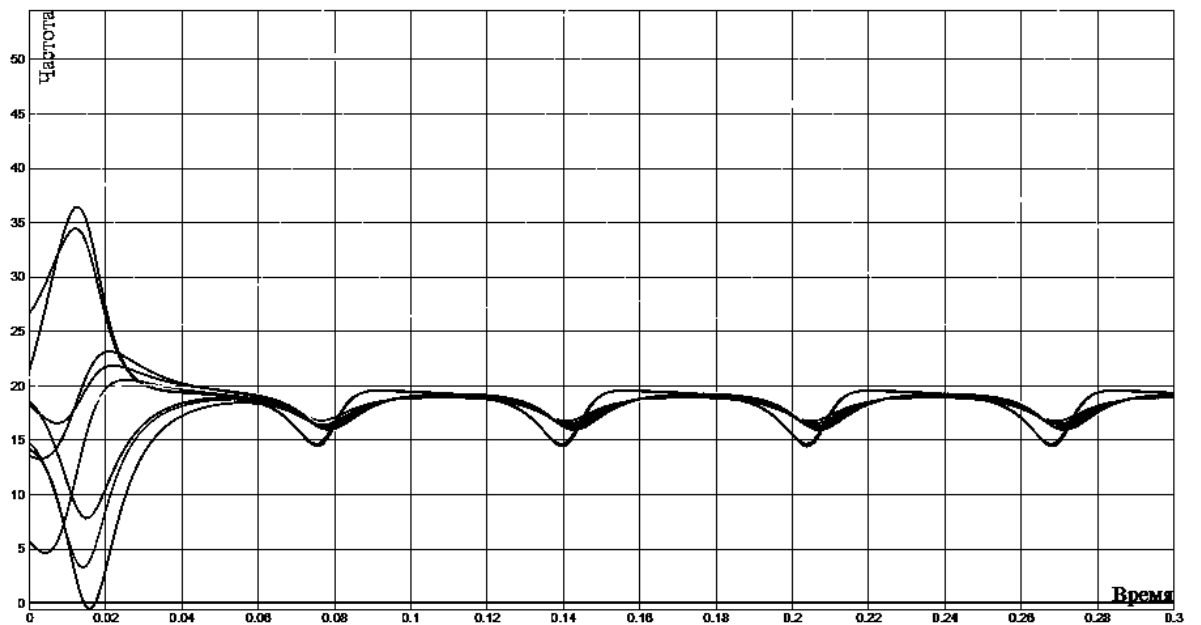


Рисунок 6 – график, получившийся осцилляторной нейронной сети, после модифицирования входных данных

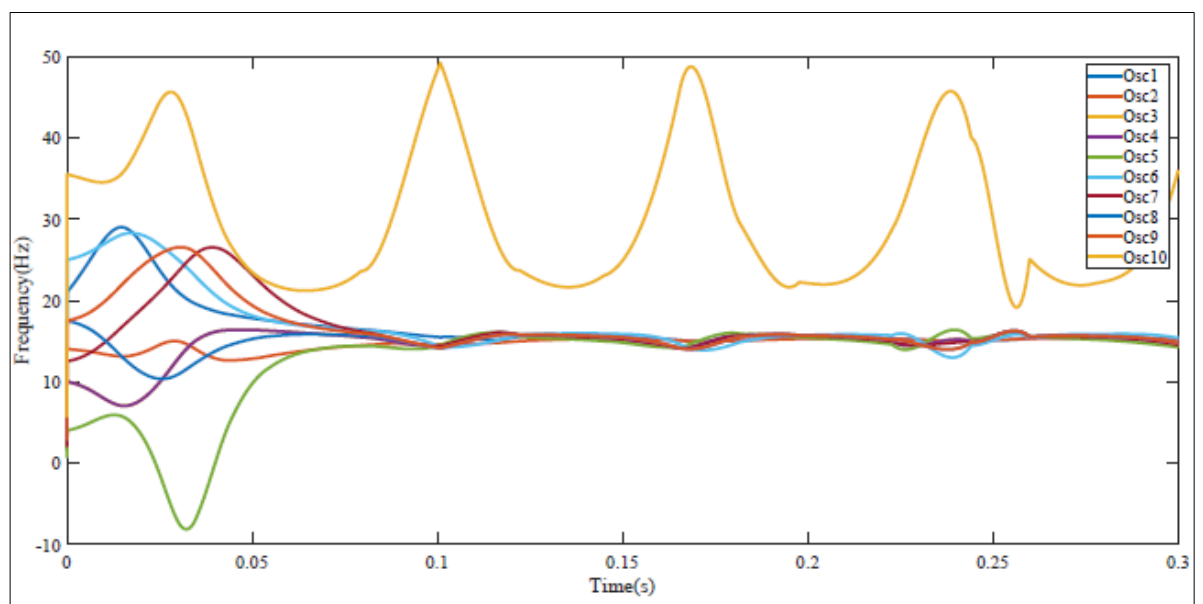


Рисунок 7 – график из статьи [13]

### 3.2. Движение автомобиля

Для реализации поставленной задачи требуется реализовать программу с трассой и движущейся по ней машины. Для последующего тестирования требуется возможность добавления пользовательского управления.

Для решения этих и последующих задач было решено реализовать класс Car, который хранит необходимую информацию о состоянии автомобиля, а именно его позицию в пространстве и направление движения (приложение Б). Позиция хранится в декартовых координатах, а направление движения в полярных. Это обусловлено тем, что в таком формате изменение направления движения будет осуществляться прибавлением соответствующих значений к числу, отвечающему за размер скорости (или же радиус в полярных координатах).

Изменение позиции машины происходит каждую единицу времени и состоит из четырех действий:

1. Влияние сопротивления на скорость;
2. Влияние ускорения на скорость;
3. Влияние поворота на угол направления;
4. Обновление позиции автомобиля.

Рассмотрим каждый этап подробно.

1. Влияние сопротивления на скорость.

Было замечено, что в силу погрешности вычислений и большого количества вычитаний значительно накапливается ошибка, что приводит к тому, что машина вместо полного замедления начинает незначительно колебаться в точке остановки. Чтобы устранить данную проблему было решено изменять значение скорости способом показанным на рисунке 8.

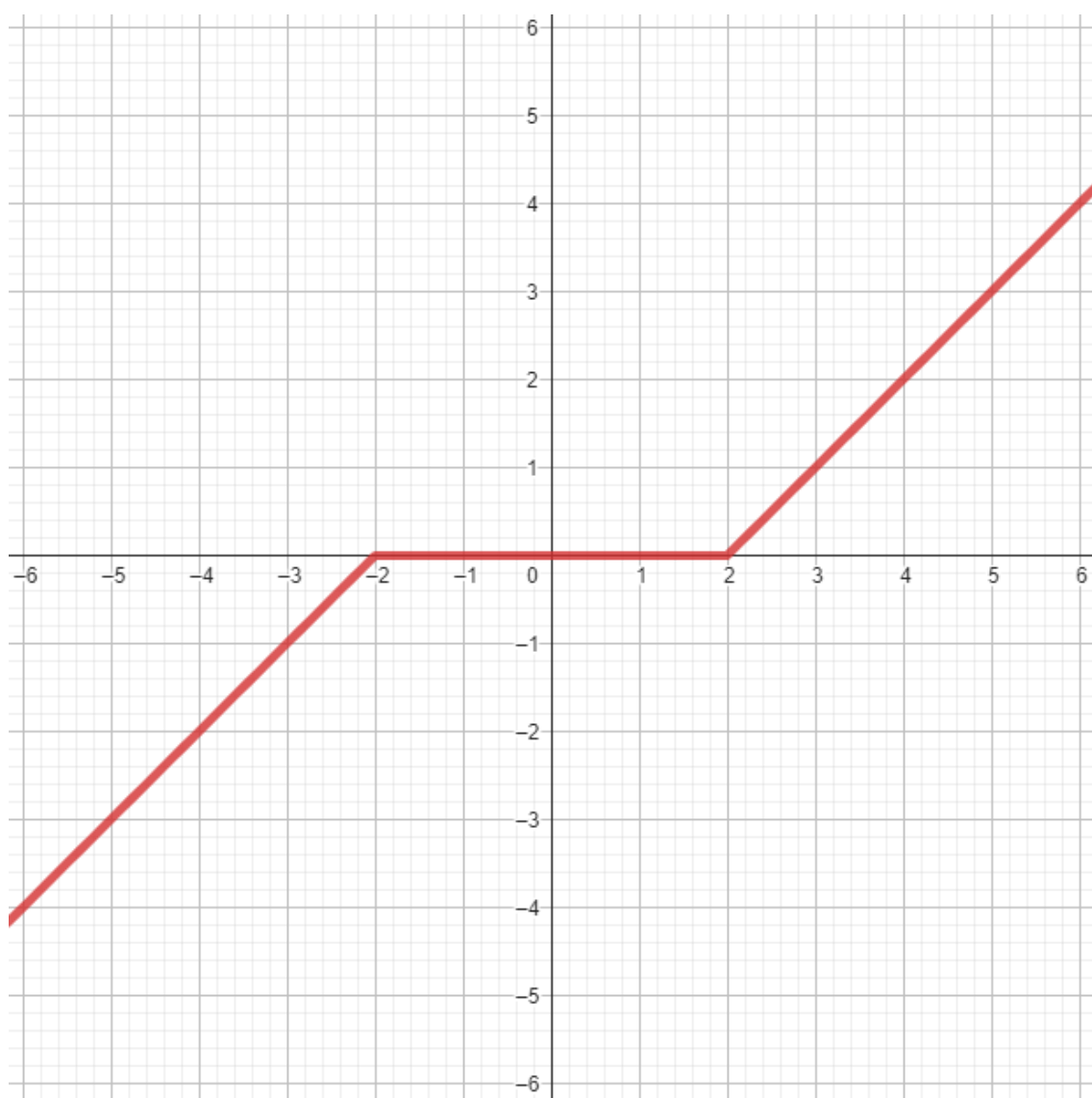


Рисунок 8 – изменение графика скорости в момент времени при силе сопротивления равным двум: по оси X – значение скорости до изменения, по оси Y – после

То есть, если график скорости по модулю больше силы сопротивления, то значение модуля уменьшается на силу сопротивления, в ином случае – значение скорости становится равным нулю.

Графики, которые задают все три отрезка – это  $s' = |s - r|$ ,  $s' = 0$  и  $s' = |s + r|$  соответственно, где  $s'$  - значение скорости  $s$  после изменения на сопротивление  $r$ . Формула, которая соответствует данному графику, выглядит так

$$s' = \frac{|s - r| - |s + r|}{2} + s \quad (8).$$



## 2. Влияние ускорения на скорость.

К значению скорости прибавляем произведение множителя направления (где  $-1$  «полный назад», а  $+1$  «полный вперед»), размера временной единицы и максимально возможное ускорения (листинг 1). Также не забываем контролировать тот факт, что скорость по модулю не может быть больше максимальной.

---

```
$step(t: number) {  
  // Сопротивление  
  const s = this.speed;  
  const r = this.options.resistance * t;  
  this.speed = (Math.abs(s - r) - Math.abs(s + r)) / 2 + s;  
  // Направление  
  if (this.direction) {  
    this.speed = between(  
      this.speed + this.getDirection() * this.options.dspeed * t,  
      this.options.maxSpeed,  
    );  
  }  
  // Поворот  
  if (this.turn) {  
    this.angle += Math.sign(this.speed)  
      * Math.abs(this.speed / this.options.maxSpeed) ** 0.5  
      * this.getTurn()  
      * this.options.turn * t;  
  }  
  // Позиция  
  this.position.$move(Vector.Polar(this.speed * t, this.angle));  
  return this;  
}
```

---

Листинг 1 — обновление состояние автомобиля

## 3. Влияние поворота на угол направления.

Так как мы хотим усложнить поведение автомобиля, сделать его не линейным и ближе приблизить к реальности, то нельзя изменять значение поворота вектора скорости прямо пропорционально значению скорости или не завесить от неё. Иначе радиус поворота не будет меняться в зависимости от значения скорости или вообще позволит вращаться автомобилю вокруг своей оси будучи неподвижным. Вследствие чего было решено использовать формулу, которая имеет квадратичную зависимость от скорости, а именно

$$a' = a + \text{sign } s \sqrt{\left| \frac{s}{\max S} \right|} t \quad (9)$$

где  $a$  – угол поворота относительно основной оси,  $\max S$  – максимальное значение скорости,  $t$  – коэффициент силы поворота в единицу времени.

#### 4. Обновление позиции автомобиля.

К точке позиции автомобиля в пространстве прибавляем вектор, который задан в полярных координатах основываясь на значении скорости и углу направления.

### 3.3. Отрисовка сцены

Имея реализацию логики движения автомобиля, необходимо научиться его отрисовывать. Для этого используется встроенный в браузерную страницу формат отрисовки через canvas. Он поддерживает такой необходимый функционал как перемещение и поворот сцены, отрисовка линий и прямоугольников.

Однако потребовалось реализовать собственную небольшую надстройку для предоставленного API для упрощения и систематизации функционала. Так, например была реализована функция, которая позволяет отрисовывать заданный объект (в частности прямоугольник) с вращением относительно заданной точки. Данные, которые мы знаем об автомобиле как раз, и заключаются в том, что мы знаем его координату в пространстве и направление движения (приложение В).

Для обновления сцены используется нативное API предоставляемое языком JavaScript, которое позволяет вызывать функцию после отрисовки браузером страницы.

Стоит обратить внимание, что для определения того, куда повернут автомобиль в пространстве – в направление его движения на передней грани изображены два прямоугольника, которые символизируют фары.

### 3.4. Обработка пользовательского ввода

Для последующей отладки было принято решение реализовать возможность управления машины пользователем, что подняло вопрос о том, как корректнее обрабатывать проверку нажатия кнопок. Было решено не подписываться несколько раз на соответствующие события от браузера, а реализовать собственную единую точку регистрации нажатий кнопок, чтобы во время разработки это было максимально удобно. Был создан единый регистратор учета нажатий кнопок, который сохранял ID всех нажатых и удалял ID отпущенных кнопок. Также был добавлен словарь всех интересующих клавиш в клавиатуре (приложение Г).

### 3.5. Создание трассы

На данный момент было рассказано о логике движения, отрисовки и управления автомобилем. Далее будет описан процесс создания трассы и обработки столкновений машины с ограждением.

Для создания пути были рассмотрены два варианта его возможной генерации: по средствам занесения вручную координат всех точек или же создание промежуточной программы, которая бы позволила нарисовать ограду трассы.

Первый вариант не требует дополнительных затрат по написанию кода, но является на порядок более ресурсозатратным, чем обработка кликов по сцене с последующей отрисовкой пути в единый замкнутый многоугольник, который будет служить оградой трассы.

Интересным моментов является невозможность напрямую узнать у canvas куда был произведен клик в нем, если считать его левый верхний угол разметки за ноль. Пришлось высчитывать координаты клика по области как разницу между абсолютными координатами в окне и положением canvas внутри рабочей зоны.

Для упрощения редактирования была дополнительно реализована функция удаления последней поставленной точки по средствам зажатия клавиши Ctrl.

После того как многоугольник, удовлетворяющий условиям, был отрисован, его координаты были сохранены. Он был добавлен в поток подрисовывания и был построен ещё один многоугольник, представляющий из себя вторую стену ограды трассы.

Сама трасса представляет из себя два полигона, массив координат которых сохранен в памяти программы и символизирует собой левую и правую ограды.

После добавления отрисовки трассы в основной поток рендера, была оставлена возможность пользователю самому задать трассу через включение флага «click» в значение «path» и передачу значения в переменную window.paths.

### 3.6. Обработка коллизий

Для проверки первого критерия качества прохождения трассы машиной, заявленного в формулировке задачи, а именно – отсутствие столкновений машины с трассой необходимо уметь определять факт того касается ли автомобиль ограды. Для решения данной проблемы было решено на основе имеющихся данных об автомобиле научиться переводить его в массив отрезков, которые бы символизировали его борта и уже в дальнейшем проверять факт коллизии этих самых отрезков с отрезками многоугольника ограды.

Чтобы не допустить ошибки в расчетах был выбран путь написания кода через максимальное обобщения и повторного использования кода с минимальным дублированием его отдельных частей (листинг 2).

---

```
const x = w / 2;
const y = h / 2;

const moveToCar = (point: SPoint) => new Point(...point)
    .$rotate(car.angle)
    .$move(car.position);

const carPoints = ([
    [x, y],
    [x, -y],
    [-x, -y],
```

```

    [-x, y],
  ] as SPoint[]).map(moveToCar);

const carCollision = carPoints
  .map((point) => point.toArr())
  .map((point, i, arr): SLine => [point, at(arr, i + 1)]);

const createLinesWithClosePath = (path: SPoint[]) => path
  .map((point, i, arr): SLine => [point, at(arr, i + 1)]);

const checkCollisionByLines = (line: SLine) => carCollision
  .some((carLine) => intersects(carLine, line));

const allCheckCollisionLine = [
  left, right,
].flatMap(createLinesWithClosePath);

collision = allCheckCollisionLine.filter(checkCollisionByLines);

```

---

## Листинг 2 — реализация проверки коллизий машины и трассы

Так сначала машина представляется как находящаяся в начале координат точка и относительно её габаритов находятся четыре её угла, координаты которых равны половине ширины и длины с чередующимися знаками. Далее для создания отрезков происходит проход по массиву, где мы берем две соседние точки (предварительно расположив их в порядке обхода и считаем первую и последнюю точки – соседними). Таким же образом из двух массивов точек ограды создаем единый массив отрезков ограды. Остается проверить пересекается ли один из отрезков автомобиля с отрезком ограды через собственную функцию проверки пересечения отрезков (листинг 3).

---

```

export const intersects = (
  [[x11, y11], [x12, y12]]: SLine,
  [[x21, y21], [x22, y22]]: SLine,
): SPoint | null => {
  const det = (x12 - x11) * (y22 - y21) - (x22 - x21) * (y12 - y11);
  if (det === 0) {
    return null;
  }
  const lambda = ((y22 - y21) * (x22 - x11) + (x21 - x22) * (y22 - y11)) / det;
  const gamma = ((y11 - y12) * (x22 - x11) + (x12 - x11) * (y22 - y11)) / det;
  if ((0 < lambda && lambda < 1) && (0 < gamma && gamma < 1)) {
    return [
      x11 + lambda * (x12 - x11),
      y11 + lambda * (y12 - y11),
    ];
  }
};

```

```
}  
    return null;  
};
```

---

### Листинг 3 — реализация коллизии отрезков

После чего мы знаем не только пересекает ли машина ограду, но и какая конкретно ограда пересечена. В начале разработки для обозначения факта коллизии происходила перекраска машины и пересеченной ограды, а в дальнейшем – автомобиль возвращался на старт.

## 3.8. Получение данных для нейронной сети

Закончился разбор функциональности, реализовывающий напрямую логику управления, изменения, отрисовки и пересечения машины и ограды. Далее перейдем к рассмотрению этапа, предшествующему интеграции нейронной сети в управление автомобилем, а именно подготовке получения и обработки входных и выходных сигналов.

Для того, чтобы автомобиль мог ориентироваться в пространстве было предложено реализовать систему лучей, исходящих от него. В реальной жизни это можно сравнить с парктроником, который посылает сигнал подобный сигналу летучей мыши и получая отраженные волны, способен определить расстояние от автомобиля до преграды.

В момент разработки использовались те же принципы, что были заложены при написании логики обработки коллизий машины с оградой, ведь, по сути, проверка пересечения луча со стеной – та же проверка коллизий за исключением некоторых пунктов, а именно:

1. Происходит другая реакция на факт пересечения;
2. Нас интересует не только факт, но и знание о том с каким отрезком луч столкнулся раньше;
3. Расчет расстояния от точки старта луча до его пересечения для последующей передачи данных в нейронную сеть.

Рассмотрим интересные пункты в хронологическом для объяснения порядке.

3. Расчет расстояния от точки старта луча до его пересечения для последующей передачи данных в нейронную сеть.

Именно с целью получения точки пересечения в реализации проверки коллизии двух отрезков возвращается объект-точка (листинг 3).

2. Нас интересует не только факт, но и знание о том с каким отрезком луч столкнулся раньше.

Зная координату точки пересечения, не составляет труда определить расстояние до точки старта луча. Однако нас интересует именно ближайшее значение, поэтому, например в данном случае невозможны некоторые оптимизации. Так, например оптимизация, которая была реализована ранее с коллизией ограды и машины, когда при нахождении первого пересечения мы совершаем выход из функции. Теперь при такой стратегии мы не смогли бы гарантировать, что точка пересечения не является ближайшей и не существует отрезка, который находится ближе.

С другой стороны, появляется другая реализованная оптимизация – оперирование в расчетах не длиной, а квадратом её значения, чтобы лишний раз не вычислять корень. Основная логика работы зрения автомобиля представлена ниже в листинге 4.

---

```
const depthEye = 200;

carEyes = ([
  [[x, 0], 0],
  [[x, y], Math.PI / 12],
  [[x, y], Math.PI / 4],
  [[x, y], Math.PI / 2],
  [[-x, y], 3 * Math.PI / 4],
  [[-x, 0], Math.PI],
] as Eye)
  .flatMap((data): Eye => (data[0][1] === 0 && data[1] % Math.PI === 0
    ? [data]
    : [data, [[data[0][0], -data[0][1]], -data[1]]]))
  .map(([data, a]): SLine => {
    const p1 = moveToCar(data);
    const p2 = p1.clone().$move(Vector.Polar(depthEye, a + car.angle));
    return [p1.toArr(), p2.toArr()];
  });
```

```

carEyesCollision = carEyes.map((line) => allCheckCollisionLine.reduce<
  [NonNullable<ReturnType<typeof intersects>>, number] | null
>((ans, oLine) => {
  const inter = intersects(line, oLine);
  if (inter) {
    const l2 = length2(line[0][0], line[0][1], ...inter);
    if (!ans || l2 < ans[1]) {
      return [inter, l2];
    }
  }
  return ans;
}, null));

carEyesCollision.forEach((data, i) => {
  // ...
  if (!data) {
    neuralValueInput[i] = 0;
    return;
  }
  const [point, l2] = data;
  neuralValueInput[i] = 1 - Math.sqrt(l2) / depthEye;
  // ...
});

```

---

#### Листинг 4 — реализация зрения автомобиля

Получается, что в первую очередь создаются сами лучи-глаза, причем для уменьшения копирования констант используется логика дублирования – если луч находится с одной стороны машины и можно создать симметрично ещё один, то он и создается. Луч задаётся как точка на автомобиле относительно начала координат и угол поворота луча относительно основной оси. Далее координаты сдвигаются в реальное положение автомобиля в пространстве и создаются отрезки, посредством отложения от уже перемещенной точки на угол, соответствующий сумме поворотов луча и поворота машины в пространстве на длину равную глубине зрения.

Если получившиеся отрезки пересекаются с отрезками ограды, то вычисляется квадрат расстояния от старта луча до пересечений и ищется такая точка, расстояние до которой минимально. Она и является минимальным расстоянием машины до ограды в заданном направлении.

Если такой точки не нашлось, то считаем что на вход нейрону поступит ноль, а чем ближе, тем прямо пропорционально больше значение, подаваемое на



нейрон вплоть до единицы (которая получается при столкновении машины с оградой).

Получаем, что на вход нейронной сети приходит десять значений расстояний от разных точек автомобиля до ограды. Заметим, что одиннадцатым значением, которое поступает на вход, нейронной сети является скорость автомобиля в данный момент времени, где полная остановка символизирует ноль, а максимальная скорость – единицу.

Для обработки выходных данных потребовалось незначительно переработать код, а именно переделать логику информирования о том нажат газ или тормоз, происходит поворот налево или направо. Ведь когда мы работаем с клавиатурным вводом – кнопка либо нажата, либо не нажата, а для данного поведения подойдут и хранение констант с перечислением. Так, например для управления ускорением использовались 3 константы («back», «forward» и «stop» – «движение назад», «движение вперед» и «остановка» соответственно) и состояние null. Для управления поворотами – 2 константы («left» и «right» – «поворот влево» и «поворот вправо» соответственно) и null.

Однако критических исправлений не последовало – строковые константы были заменены на числовые константы, где крайними состояниями являлись –1 и +1 (листинг 5).

---

```
getDirection(): number {
  switch (this.direction) {
    case 'back':
      return -1;
    case 'forward':
      return 1;
    case 'stop':
      return -Math.sign(this.speed);
    case null:
      return 0;
    default:
      return this.direction;
  }
}

getTurn(): number {
  switch (this.turn) {
```

```

        case 'left':
            return -1;
        case 'right':
            return 1;
        case null:
            return 0;
        default:
            return this.turn;
    }
}

```

---

Листинг 5 — реализация определения множителя поворота и ускорения в зависимости от поступающего сигнала

Подытоживая вышеописанное – от машины нейронная сеть будет получать 11 значений – 10 расстояний от лучей-глаз и скорость транспортного средства, а на выход для управления автомобилем будет выдавать 2 – силу ускорения и направление поворота.

### 3.9. Создание обучающей выборки

На этапе разработки уже был затронут факт того, что создать данную выборку не составит особого труда, однако не затронуть данный аспект реализованной программы не представляется возможным так как качество и объем обучающей выборки напрямую влияет на качество получившейся нейронной сети.

Для записи данных потребовалось реализовать систему проверки прохождения трассы, поэтому по всему пути были расставлены полосы-маркеры, пересечение с которыми увеличивает счетчик и когда он достигает отметки равной количеству этих меток круг считается пройденным и позиция автомобиля сбрасывается для нового заезда.

Для сохранения обучающей выборки каждый рендер данные о состоянии входных и выходных нейронах сохраняются в промежуточный массив. Если попытка признана не удачной, то массив сбрасывается, а если путь проходит успешно, то добавляется в глобальный массив успешных заездов и его значение

пишется в консоль браузера для того, чтобы иметь возможность сохранить его в json файл для последующего обучения.

Для генерации обучающей выборки осталось качественно и быстро проехать круг «вручную» необходимое количество раз.

### 3.10. Логика пользовательского интерфейса

Исходя из поставленной задачи дипломной работы пользователь должен иметь возможность:

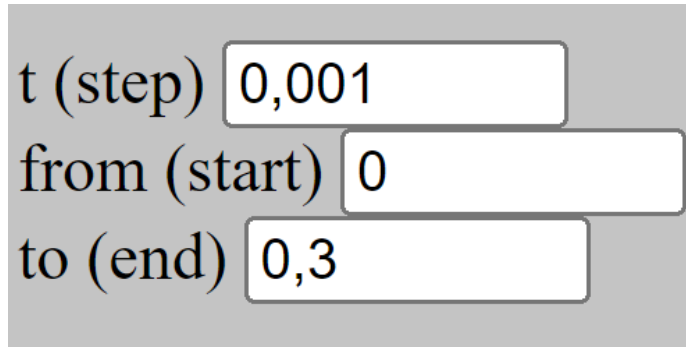
1. Воспроизвести эксперимент, который проверит корректность работы нейронной сети;
  2. Создать обучающую выборку по средствам прохождения трассы в ручном режиме и с учетом вида зрения, который будет предоставлен нейронным сетям;
  3. Запустить процесс обучения нейронной сети и в случае успешного её прохождения – сохранить;
  4. Загрузка ранее обученных нейронных сетей для последующего анализа.
- Рассмотрим каждый пункт подробнее.

1. Воспроизвести эксперимент, который проверит корректность работы нейронной сети.

Пользователь может указать следующие значения для воспроизведения эксперимента:

1.  $t$  – размер минимального временного шага, на который будет происходить смещение при перерасчете значений осцилляторной нейронной сети;
2. *from* и *to* – левая и правая временные границы соответственно. Они отвечают за то, какая часть графика будет изображена на экране после генерации. По умолчанию данные значения совпадают со значениями из эксперимента (рисунок 9).

3. contrast – флаг запуска, указываемый в ссылке. Позволяет сделать вывод графика контрастным. По умолчанию выключен и графики колебаний выделены каждый своим цветом. Был добавлен для удобства демонстрации работы программы.



t (step) 0,001  
from (start) 0  
to (end) 0,3

Рисунок 9 – форма ввода данных для графика со значениями по умолчанию

Пользователь может сгенерировать заново графики для получения новых данных. Так как за стартовые значения состояние осцилляторов в осцилляторной нейронной сети, а именно за начальное состояние фазы, отвечает генератор случайных последовательностей (рисунок 10). Это было сделано именно таким образом так как в статье [13] начальные значения не уточнялись.

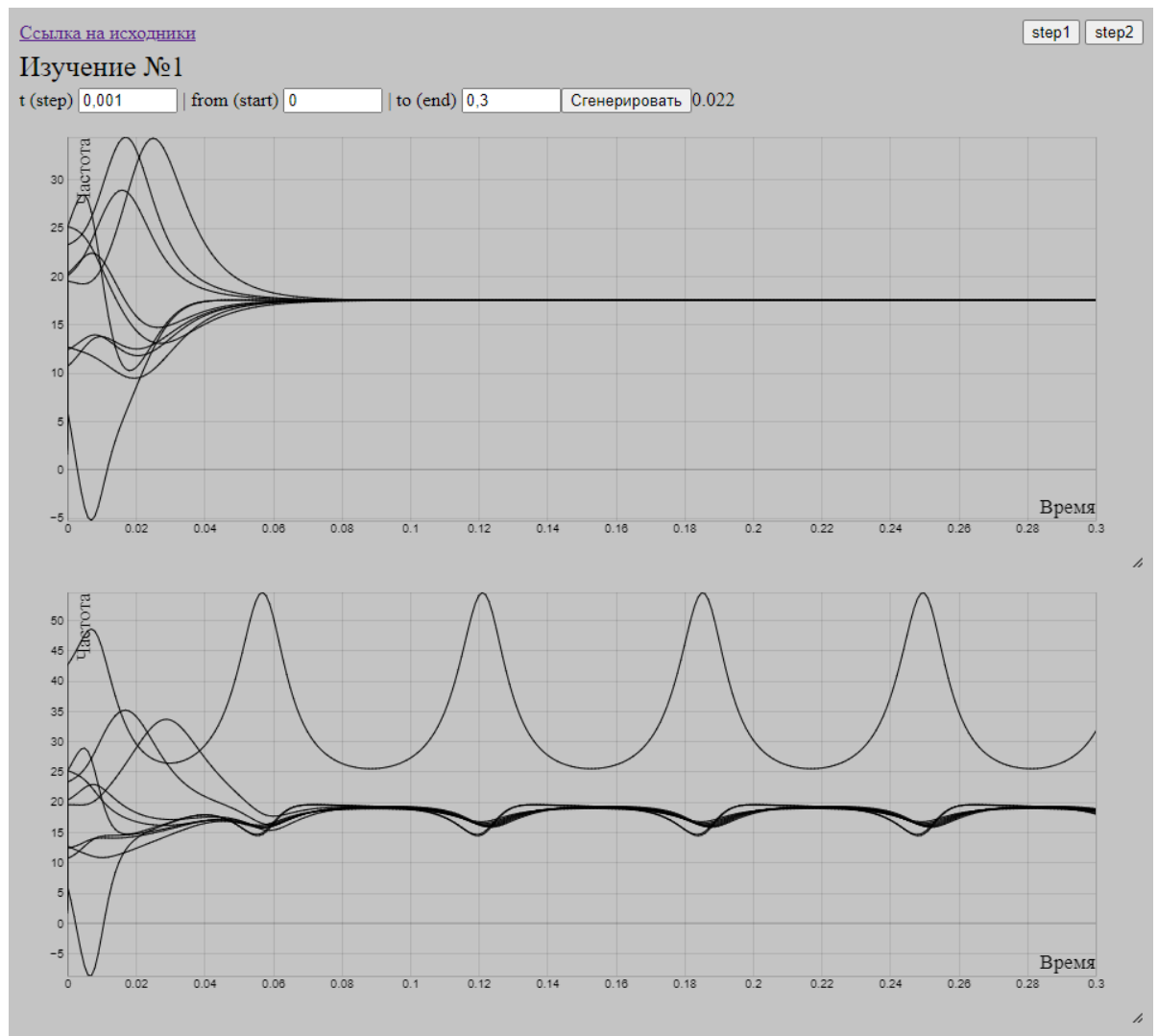


Рисунок 10 – пользовательский интерфейс анализа корректности работы  
ОНС

Отдельно стоит подчеркнуть возможность пользователя отслеживать время, затраченное на вычисление данных и рендер графика, что позволяет в случае дальнейшего анализа, путем изменения вводимых данных, точнее оценивать требуемое время для воспроизведения эксперимента.

Для удобства использования была добавлена возможность динамически изменять размеры графиков и синхронизация позиций рассматриваемых областей. Это значительно упрощает взаимодействие пользователя с интерфейсом.

2. Создать обучающую выборку по средствам прохождения трассы в ручном режиме и с учетом вида зрения, который будут предоставлен нейронным сетям;

Пользователь имеет возможность настроить окружение взаимодействия автомобиля под себя. Ему доступны следующие параметры для визуализации и создания обучающей выборки:

1. Переключение состояния показа трассы;
2. Переключение состояния показа контрольных точек;
3. Переключение состояния показа лучей-глаз автомобиля.

Данный набор настроек позволяет не только провести проверку корректности алгоритма работы зрения машины, но и оставить себе только тот вид что имеется у транспортного средства. Это позволяет в равной степени и корректно обучить, и осознать какие данные не доступны (рисунки 11 и 12).

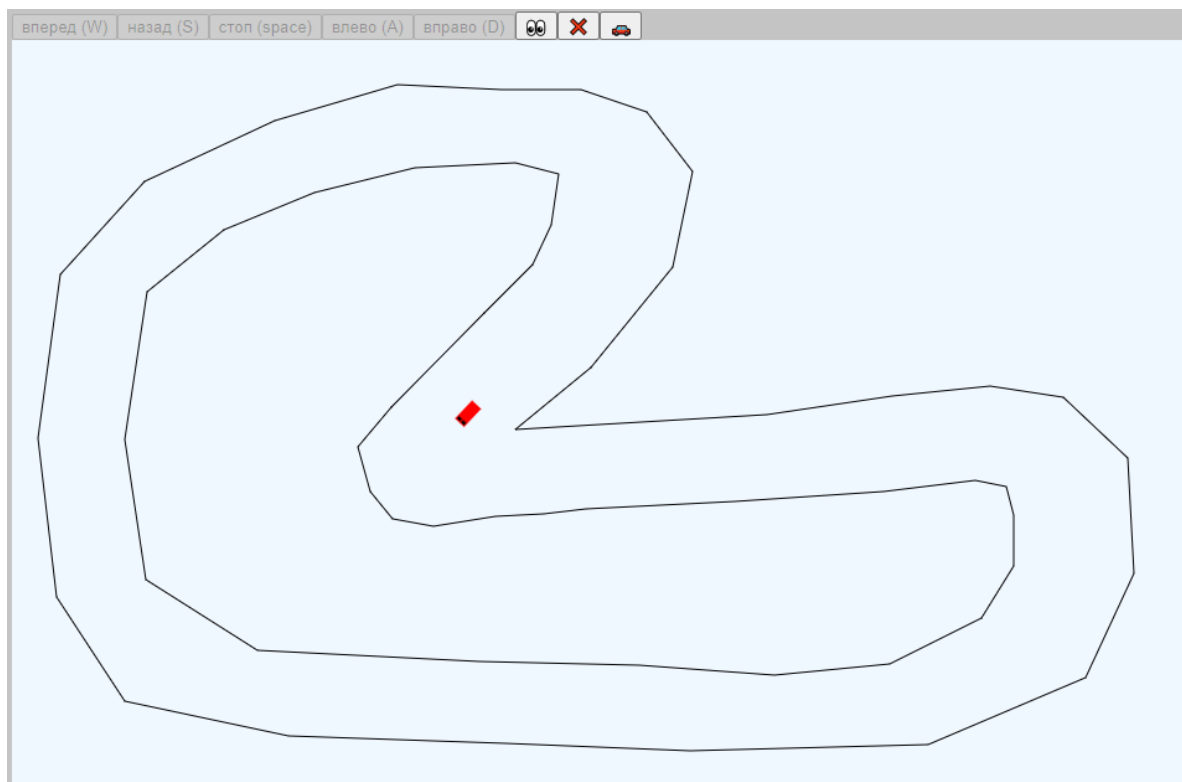


Рисунок 11 – вид прохождения маршрута со включенной оградой и  
выключенным показом лучей-глаз

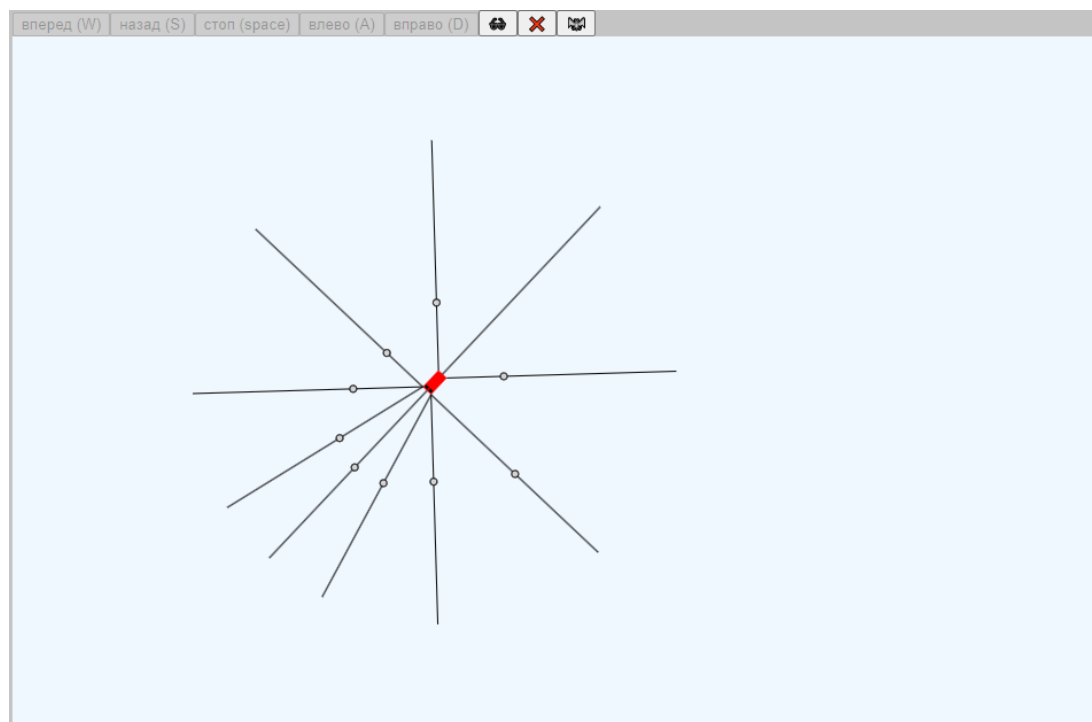


Рисунок 12 – вид прохождения маршрута, с позицией аналогичной изображенному на рисунке 11, но с выключенной оградой и включенным показом лучей-глаз

Во время управления движением автомобиля по трассе посредством клавиатуры над моделью изображено какие кнопки в данный момент нажаты (рисунок 13).

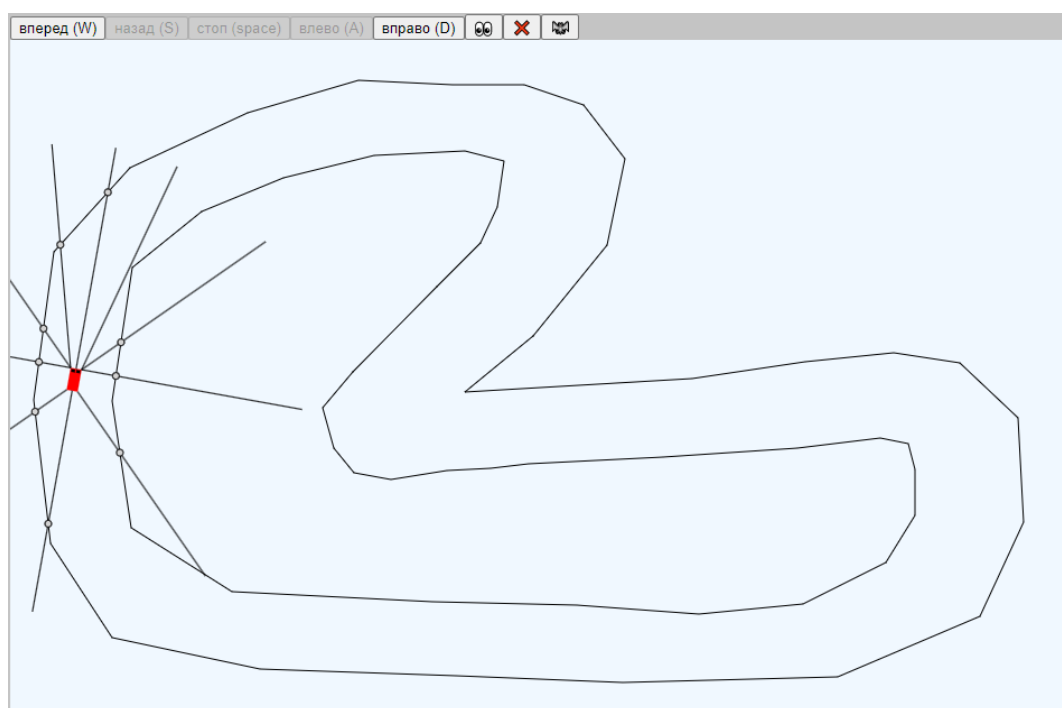


Рисунок 13 – пример нажатия клавиш вперед и вправо

После прохождения круга в консоли браузера будет выведен массив с обучающей выборкой. Если пользователь считает, что она недостаточно велика, то может продолжить проходить трассы – данная выборка будет сохраняться и пополняться после успешного прохождения каждого круга.

3. Запустить процесс обучения нейронной сети и в случае успешного её прохождения – сохранить.

Пользователь посредством добавления хеша ссылки включает режим обучения. Для конфигурирования обучения ему доступны следующие опции:

1. Переключение вида обучаемой нейронной сети – классической или осцилляторной (флаг «`onp`», по умолчанию – выключен, выбрана классическая);

2. Коэффициент обучения нейронной сети (флаг «`rate`», по умолчанию равен 0.2);

3. Коэффициент умножения количества итерация обучения – всего итераций 2000 (флаг «`iter`», по умолчанию множитель равен 1, то есть итераций обучения будет в точности 2000);

4. Количество нейронов на скрытом слое у выбранной нейронной сети (флаг «`hiddenLayers`», по умолчанию – 6).

После конфигурирования значений, включения обучения (флаг «`train`») и применения их путем обновления страницы запускается обучение. Последующий алгоритм действия программы выглядит следующим образом:

1. Нейронная сеть начинает обучение в фоновом режиме, автомобиль находится на месте, при желании в моменты ожидания есть возможность в ручном режиме пройти трассу.

2. После окончания обучения позиция автомобиля сбрасывается в исходное состояние, и обучившаяся нейронная сеть берет на себя управление автомобилем.

3. Если автомобиль успешно проходит круг, то он продолжит двигаться по трассе, а у пользователя будет возможность через консоль сохранить состояние получившейся нейронной сети.



4. Если автомобиль врывается в ограду, не пройдя заданный маршрут, то нейронная сеть признается на работающей, страница перезагружается и процесс обучения стартует заново с новой нейронной сетью. Данный подход позволяет в автоматическом режиме подбирать удачные стартовые значения и изменять конфигурацию значений. Изменения хеша ссылки не приводит к обновлению страницы, а после неудачного прогона – будут применены уже измененные значения.

Для мониторинга за работой нейронной сети, в правом верхнем углу окна расположены данные:

1. Количество кадров в секунду (fps);
2. Данные от лучей-глаз (eye0-eye9). При наведении на значение соответствующий луч выделяется для контроля обработки и ориентирование в данных (рисунок 14);
3. Скорость, ускорение и поворот автомобиля;
4. Количество корректно пройденных маркеров и время прохождения трассы;

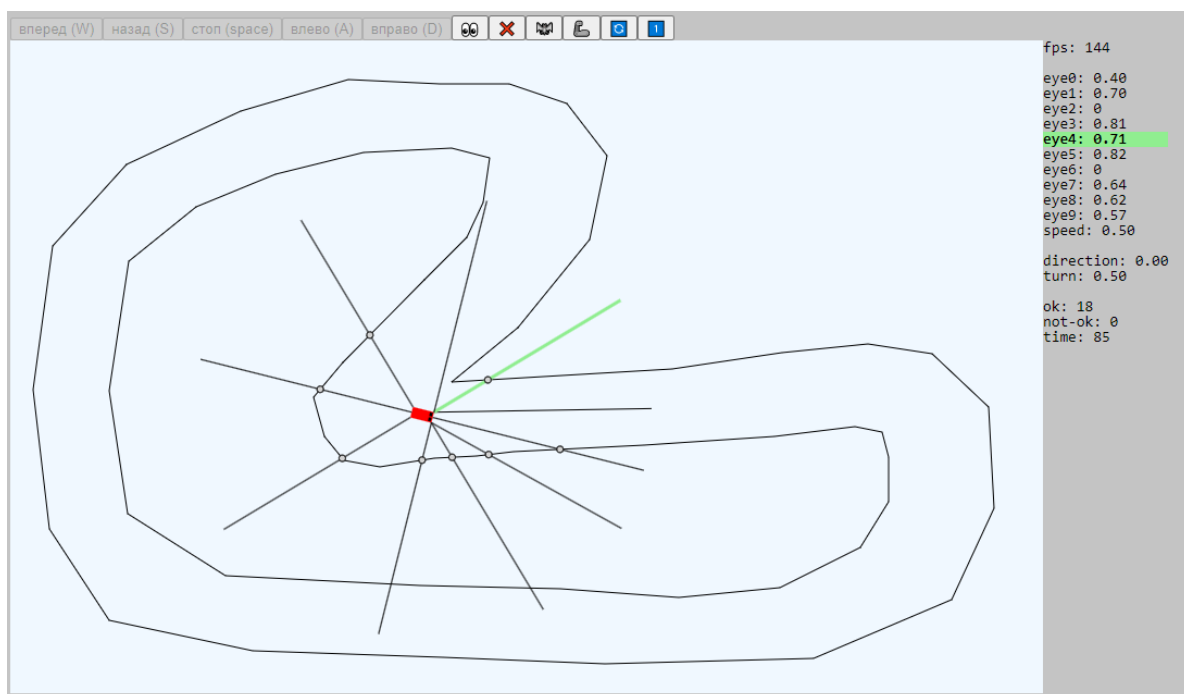


Рисунок 14 – пример отображения данных с выделенной eye4

#### 4. ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ

Тестирование проводилось на оборудовании, оснащённом процессором AMD Ryzen 7 4800H с базовой тактовой частотой 2.9 ГГц и оперативной памятью объемом 16 ГБ.

Целью тестирования было:

1. Проверка корректности работы осцилляторной нейронной сети за счет сравнения результатов его работы с результатами эксперимента, представленного в статье [13];
2. Проверка качества обучения ОНС и классической нейронной сети;
3. Выявление ошибок и отладка разработанного приложения.

Рассмотрим каждый пункт подробнее.

1. Проверка корректности работы осцилляторной нейронной сети за счет сравнения результатов его работы с результатами эксперимента, представленного в статье [13].

В данной статье есть много недосказанности, что во время работы привело к некоторым проблемам. Во-первых, стоит обратить внимания на то, что график, который получается это не график зависимости времени от фазы, а график времени от частоты (рисунок 15). Частоту же в свою очередь можно вывести как производную фазы. Для этого пришлось вынести её вычисляемое значение в отдельную переменную.

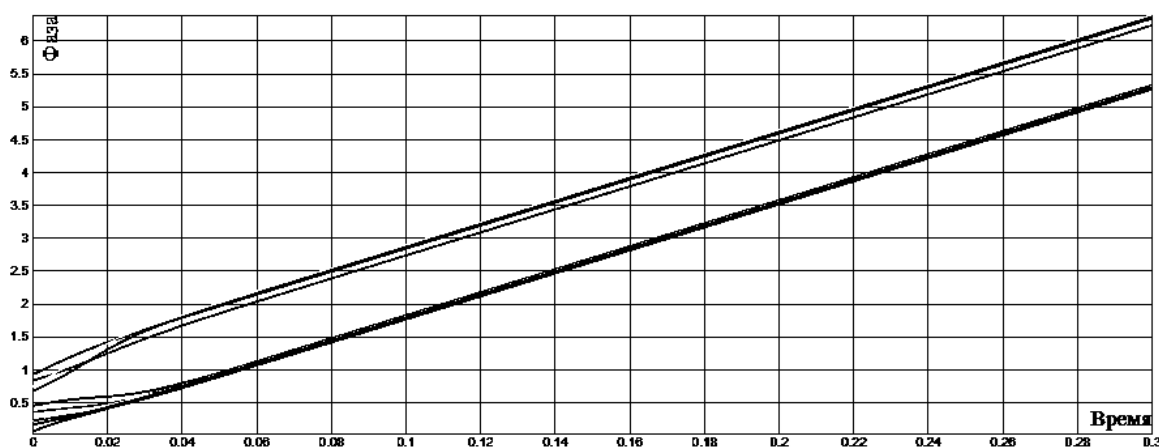


Рисунок 15 – график значения фазы от времени при заданных данных

Во-вторых, в статье [13] не упоминается, что в формулах происходит сдвиг на  $2\pi$ . При построении графика для определения частоты происходит повторный сдвиг в обратную сторону. А для оценки схожести графиков во время разработки был взят за основу тот факт, что первый график в среднем стремится к 17.5, однако без деления на  $2\pi$  этого не происходило.

## 2. Проверка качества обучения ОНС и классической нейронной сети;

Для того чтобы нейронные сети находились в равных условиях было решено использовать один скрытый слой и обучить 14 нейронных сетей с количеством нейронов на скрытом слое от 4 до 10 (7 возможных значений количества нейронов на скрытом слое и 2 виде нейронных сетей).

Во время ручного тестирования был подтвержден факт того, что тестовая выборка играет важную роль в обучении. Так, например, если вместо того, чтобы зажать и держать кнопки для движения много раз на них нажимать или проходить трассу слишком медленно, то нейронные сети обучаются вообще не двигаться и оставаться на своих позициях. Для решения этой проблемы потребовалось несколько раз пересоздать тестовую выборку.

После того как нейронные сети прошли обучение на одинаковых выборках с одинаковым количеством итераций и коэффициентами обучения, получилось так, что осцилляторная нейронная сеть в среднем проходит трассу за 11 секунд, а классическая – за 15.

## 3. Выявление ошибок и отладка разработанного приложения.

На этапе тестирования компонентов разработанного приложения были выявлены разнообразные ошибки. Так, например canvas хоть и имеет декартову систему координат, но ось Y направлена вниз и в то же время направление угла поворота происходит по часовой стрелки. Также интересным примером является отладка зрения автомобиля: оказалось, что если пересечение не происходило, то значение в массиве данных не обновлялось что приводило к тому, что в момент инициализации в массиве были не валидные для обучения данные и нейронная сеть не могла обучиться из-за единичных сломанных тестов.

Чтобы показать, что не произошло переобучения было решено добавить несколько трасс и возможность переключаться между ними (рисунки 16 и 17).

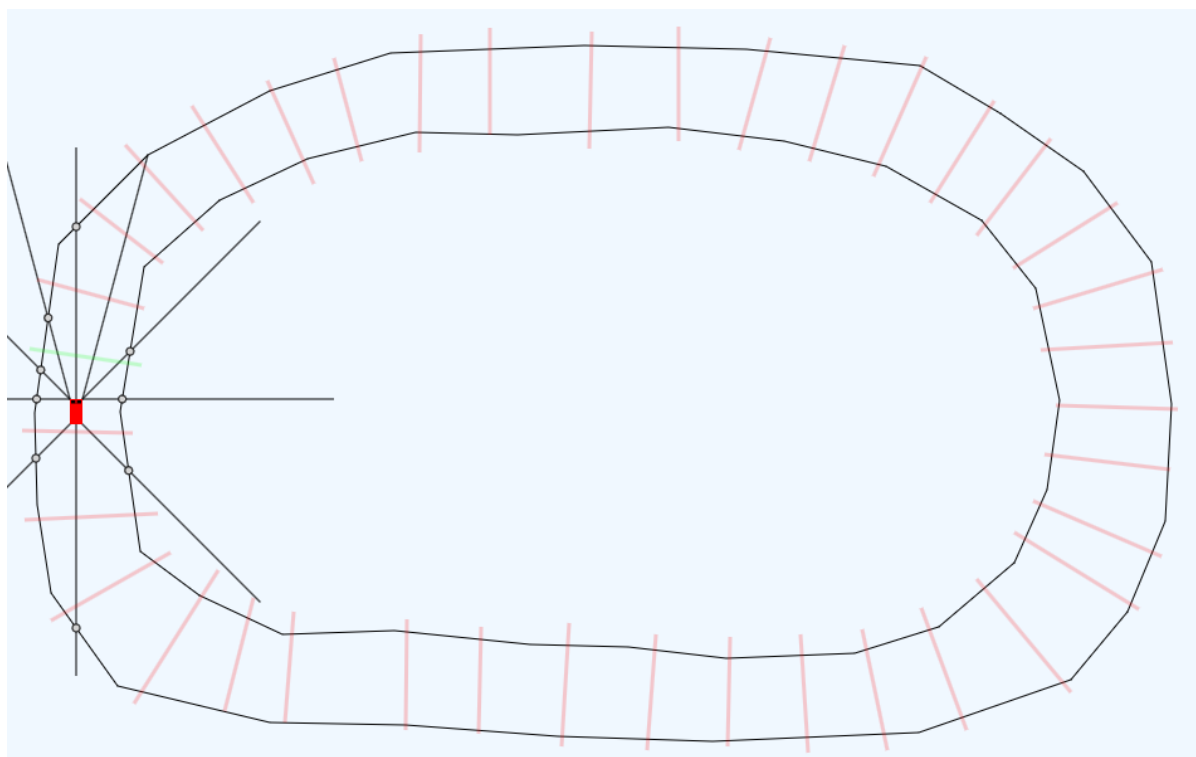


Рисунок 16 – трасса для тестирования №2

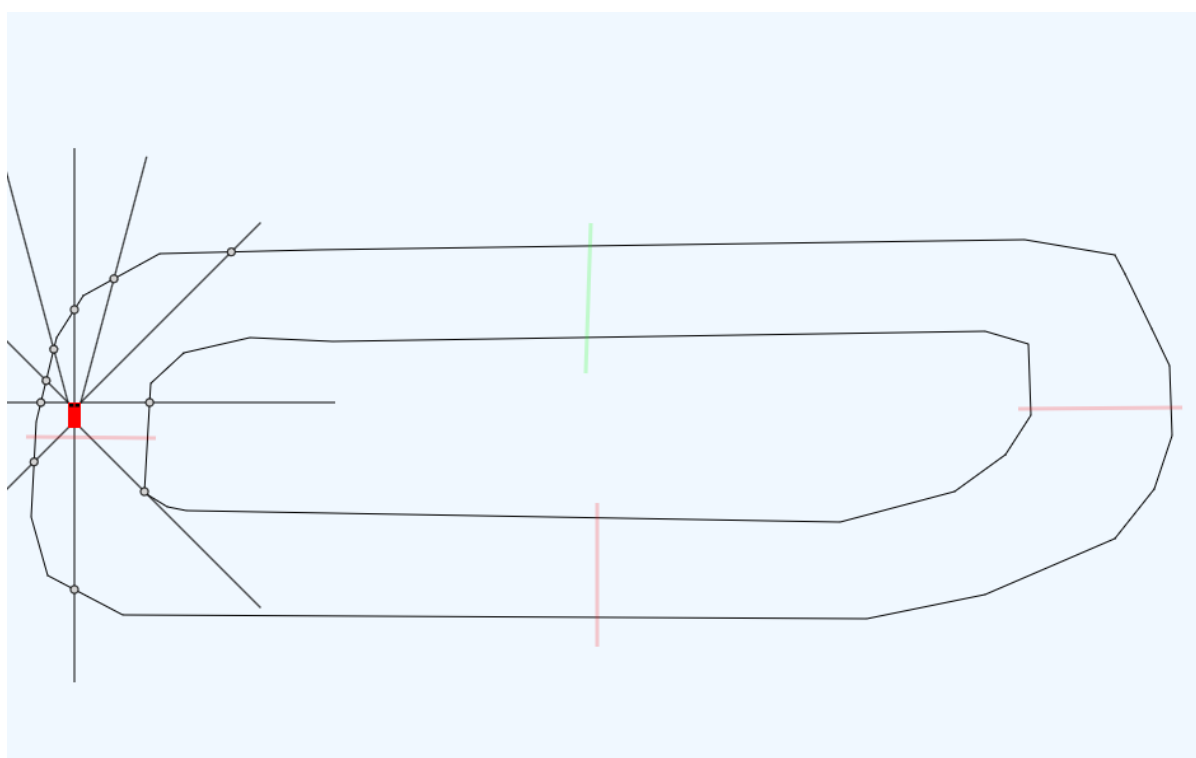


Рисунок 17 – трасса для тестирования №3

Во время тестирования работоспособности ОНС и классической нейронной сети была выявлена интересная особенность. Если случалось так, что автомобиль под управлением классической нейронной сети оказывался в ситуации, изображённой на рисунке 18, то это приводило к полной остановке машины под управлением классической нейронной сетью.

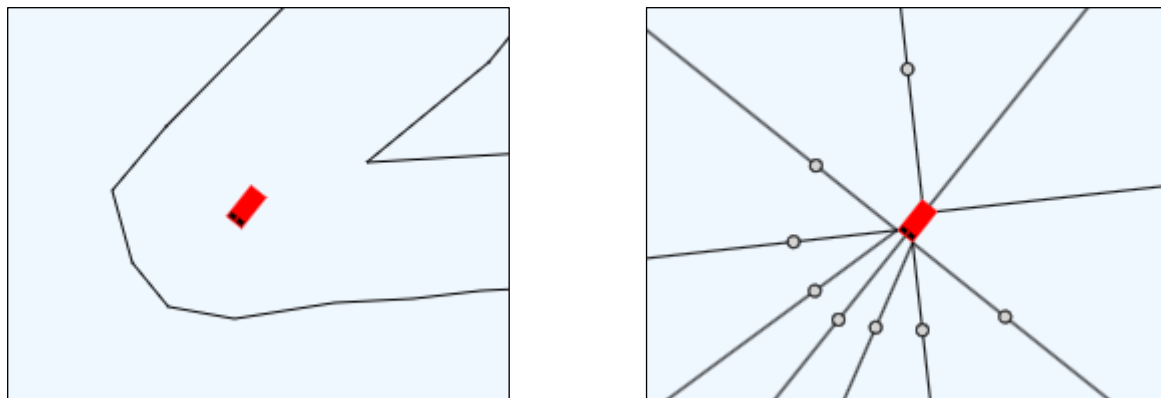


Рисунок 18 – машина на крутом повороте налево: вид от пользователя (слева) и со стороны нейронной сети (справа)

В то же время ОНС, оказавшаяся в похожей ситуации, зачастую продолжала движение. Это вызвано тем, что данные, получаемые нейронной сетью, не показывают картину целиком и ОНС имеет логику запоминания и ритмичности процесса в отличие от персептрона.

## ЗАКЛЮЧЕНИЕ

В рамках данной дипломной работы была изучена и реализована осцилляторная нейронная сеть. Был применен такой способ её изучения как изменение функции преобразования и весов связей, а также изучена корректность ОНС.

Был реализован эксперимент, предложенный в статье [13] для проверки корректности работоспособности осцилляторной нейронной сети. В ходе его работы были выявлены особенности работы ОНС такие как:

1. Результирующим значением нейрона является частота, а не фаза;
2. Начальные значения фаз нейронов не оказывают значительного влияния на конечный результат системы с течением времени.

Для решения поставленной задачи была разработана собственная модель, представляющая из себя прохождение трассы автомобилем. Было реализовано зрение для транспортного средства посредством испускания нескольких лучей и вычисление расстояния от машины до препятствия.

Было добавлено управление нейронной сетью автомобилем и логика проверки правильности прохождения пути. Было добавлено обучение нейронной сети на основе обучающей выборки и перезапуск обучения при неудачном прохождении трассы. Это помогло автоматизировать процесс обучения и сбор данных для анализа.

В ходе работы на основе полученных данных была составлена таблица 1 и основываясь на её данных был построен рисунок 19.

Таблица 1 – результаты эффективности обучения классической и осцилляторной нейронных сетей при разном количестве нейронов на скрытом слое

Кол-во нейронов на скрытом слое	Классическая нейронная сеть		Осцилляторная нейронная сеть	
	Вероятность прохождения первого круга	Вероятность прохождения второго круга	Вероятность прохождения первого круга	Вероятность прохождения второго круга
4	0,83	0,8	0	0
5	0,87	0,84	0,5	0,42

Продолжение таблицы 1

6	0,93	0,91	0,62	0,54
7	0,92	0,89	0,67	0,55
8	0,88	0,84	0,71	0,63
9	0,83	0,79	0,7	0,62
10	0,78	0,74	0,65	0,53

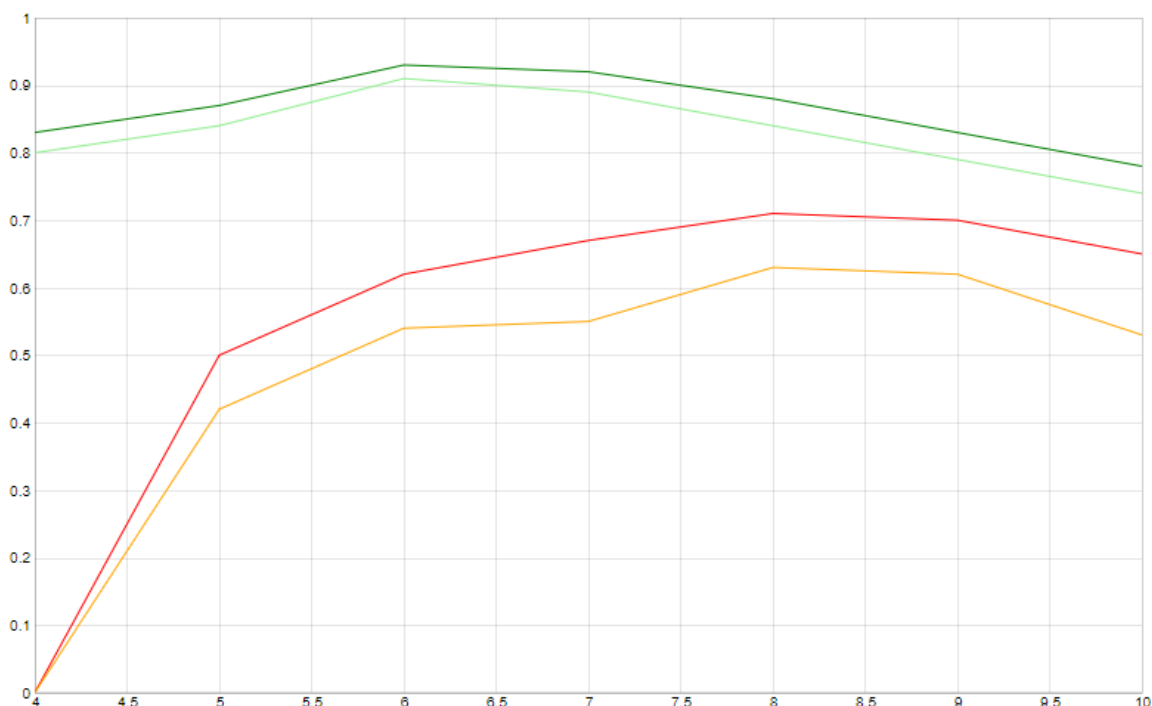


Рисунок 19 – график основанный на данных из таблицы 1

Анализ результатов показал, что осцилляторная нейронная сеть работает лучше, чем классическая нейронная сеть по обоим критерия:

1. Для прохождения автомобилем трассы требуется на 25% меньше нейронов на скрытом слое для ОНС по сравнению с классической нейронной сетью;
2. В лучших своих конфигурация ОНС проходит трассу на 20% чаще, чем классическая нейронная сеть.

Данные результаты приводят к выводу о том, что осцилляторная нейронная сеть справляется лучше примерно на 20–25%, чем классическая нейронная сеть с задачей управления и координацией в пространстве.

В дальнейшем, возможно, добавление новых способов для анализа ОНС, например усложнение системы путем добавления других автомобилей (трафика) и добавления большего количества получаемой и выдаваемой информации.



## СПИСОК ЛИТЕРАТУРЫ

1. Ларионов Д. Лекция по теме «Нейроморфные вычисления». Москва: ICPC Moscow Regional Contest. 2022.
2. Chuan Li. OpenAI's GPT-3 Language Model: A Technical Overview. Lambda Blog [Электронный ресурс] – Электрон. Текстовые дан. – URL: <https://lambdalabs.com/blog/demystifying-gpt-3>
3. Zhang L., Li Xv, Xue T. Resonant synchronization and information retrieve from memorized Kuramoto network. Adaptation and Self-Organizing Systems. 2018.
4. Баскин И.И., Палюлин В.А., Зефирова Н.С. Многослойные перцептроны в исследовании зависимостей «структура-свойство» для органических соединений. Российский химический журнал. 2006.
5. Борисюк Г.Н., Борисюк Р.М., Казанович Я.Б., Лузянина Т.Б., Турова Т.С., Цымбалюк Г.С. Осцилляторные нейронные сети. Математические результаты и приложения. Математическое моделирование. 1992.
6. Стоянов А.К. Осцилляторная нейронная сеть классификатор. Известия ТПУ. 2010.
7. Казанович Я.Б., Борисюк Р.М. Нейросетевая модель слежения за несколькими объектами. Нейроинформатика. 2006.
8. Кузьмина М.Г., Маныкин Э.А., Сурина И.И. Осцилляторная сеть с самоорганизованными динамическими связями для сегментации изображений. Математическое моделирование. 2017.
9. Федосин С.А., Ладяев Д.А., Марьина О.А. Анализ и сравнение методов обучения нейронных сетей. Вестник МГУ. 2010.
10. Brain.js API Documentation [Электронный ресурс] – Электрон. Текстовые дан. – URL: <https://github.com/BrainJS/brain.js#brainjs>
11. Function Plot API Documentation [Электронный ресурс] – Электрон. Текстовые дан. – URL: <https://mauriciopoppe.github.io/function-plot/>

12. Module path maps are not resolved in emitted code. TypeScript Issues [Электронный ресурс] – Электрон. Текстовые дан. – URL: <https://github.com/microsoft/TypeScript/issues/10866>

13. Zhang T. Haider M.R. Massoud Y. Alexander J.I.D. An Oscillatory Neural Network Based Local Processing Unit for Pattern Recognition Applications. Electronics. 2019.

## ПРИЛОЖЕНИЕ А. Реализация ОНС

В листинге представлен код, реализующий осцилляторную нейронную сеть.

```
import type { M1, M2 } from './Matrix';
import { sum } from './sum';

const TwoPi = 2 * Math.PI;

export class ONN {
  public phi: M1;

  public dphi: M1;

  public N: number;

  constructor(
    public K: M2,
    public f: M1,
    public t: number,
    initPhi?: M1,
  ) {
    this.N = f.length;

    if (![
      K.length,
      ...K.map((k2) => k2.length),
    ].every((size) => size === this.N)) {
      throw new RangeError('incorrect size');
    }

    this.phi = initPhi || [...f];
    this.dphi = [...f];
  }

  step() {
    const {
      phi, K, f, N, t,
    } = this;
    this.dphi = phi.map((_, i) => TwoPi * (
      f[i] + sum(N, (j) => K[i][j] * Math.sin(phi[j] - phi[i]))
    ));
    this.phi = phi.map((_, i) => (phi[i] + t * this.dphi[i]) % TwoPi);
  }
}
```

## ПРИЛОЖЕНИЕ Б. Реализация автомобиля

В листинге представлен код, реализующий автомобиль.

```
export class Car {
  public position: Point;

  public speed = 0;

  #angle = 0;
  get angle() {
    return this.#angle;
  }
  set angle(newAngle: number) {
    this.#angle = newAngle % (2 * Math.PI);
  }

  public direction: 'forward' | 'back' | 'stop' | number | null = null;

  public turn: 'left' | 'right' | number | null = null;

  public initPosition: Point;

  public initAngle: number;

  constructor(
    x: number,
    y: number,
    public options: {
      maxSpeed: number, turn: number; resistance: number; dspeed: number;
angle?: number;
    },
  ) {
    this.initPosition = new Point(x, y);
    this.position = new Point(x, y);
    if (this.options.angle) {
      this.angle = this.options.angle;
    }
    this.initAngle = this.angle;
  }

  $reset() {
    this.position = this.initPosition.clone();
    this.angle = this.initAngle;
    this.speed = 0;
  }

  getDirection(): number {
    switch (this.direction) {
      case 'back':
```

```

        return -1;
    case 'forward':
        return 1;
    case 'stop':
        return -Math.sign(this.speed);
    case null:
        return 0;
    default:
        return this.direction;
    }
}

getTurn(): number {
    switch (this.turn) {
        case 'left':
            return -1;
        case 'right':
            return 1;
        case null:
            return 0;
        default:
            return this.turn;
    }
}

$step(t: number) {
    // Сопротивление
    const s = this.speed;
    const r = this.options.resistance * t;
    this.speed = (Math.abs(s - r) - Math.abs(s + r)) / 2 + s;
    // Направление
    if (this.direction) {
        this.speed = between(
            this.speed + this.getDirection() * this.options.dspeed * t,
            this.options.maxSpeed,
        );
    }
    // Поворот
    if (this.turn) {
        this.angle += Math.sign(this.speed)
            * Math.abs(this.speed / this.options.maxSpeed) ** 0.5
            * this.getTurn()
            * this.options.turn * t;
    }
    // Позиция
    this.position.$move(Vector.Polar(this.speed * t, this.angle));
    return this;
}
}

```

## ПРИЛОЖЕНИЕ В. Отрисовка в пространстве полигона и автомобиля

В листинге представлен код, реализующий отрисовку полигона в пространстве с вращением вокруг любой точки (по умолчанию – вокруг центра) и отрисовка автомобиля.

```
const canvas = document.getElementById('step2-canvas') as HTMLCanvasElement;

const ctx = canvas.getContext('2d');

const fillRect = ({
  x, y, w, h, a = 0, cx = x + w / 2, cy = y + h / 2,
}: {
  x: number,
  y: number,
  w: number,
  h: number,
  a?: number,
} & AllOrNever<{
  cx: number,
  cy: number,
}>) => {
  ctx.translate(cx, cy);
  ctx.rotate(a);
  ctx.translate(-cx, -cy);

  ctx.fillRect(x, y, w, h);

  ctx.setTransform(1, 0, 0, 1, 0, 0);
};

const w = 20;
const h = 10;

const s = 300;

const car = new Car(55, 320, {
  maxSpeed: s,
  turn: Math.PI,
  resistance: s,
  dspeed: 2 * s,
  angle: -Math.PI / 2,
});

const carColor = 'red';

const drawCar = () => {
  const cx = car.position.x;
  const cy = car.position.y;
```

```

const x = cx - w / 2;
const y = cy - h / 2;
const a = car.angle;
ctx.fillStyle = carColor;
fillRect({
  x, y, w, h, cx, cy, a,
});

const miniW = w / 8;
const miniH = h / 3;

ctx.fillStyle = 'black';
fillRect({
  x: x + w - miniW - 1, y: y + 1, w: miniW, h: miniH, cx, cy, a,
});
fillRect({
  x: x + w - miniW - 1, y: y + h - miniH - 1, w: miniW, h: miniH, cx, cy, a,
});
};

```

## ПРИЛОЖЕНИЕ Г. Обработка пользовательского ввода

В листинге представлен код, реализующий системную обработку нажатий пользователем с последующим примером использования данного API.

```
const mapKeys = {
  up: 87,
  down: 83,
  left: 65,
  right: 68,
  space: 32,
};

type Keys = keyof typeof mapKeys;

export const keyController = () => {
  const keysDown: Partial<Record<number, boolean>> = {};

  const keyActive = (key: Keys) => keysDown[wasdKeys[key]] || false;

  window.addEventListener('keydown', (e) => {
    keysDown[e.which] = true;
  });

  window.addEventListener('keyup', (e) => {
    keysDown[e.which] = false;
  });

  return {
    keyActive,
  };
};

/** split file */

import { keyController } from '../controller';
const { keyActive } = keyController();

if (keyActive('space')) {
  car.direction = 'stop';
} else if (keyActive('up')) {
  car.direction = 'forward';
} else if (keyActive('down')) {
  car.direction = 'back';
} else {
  car.direction = null;
}
```