**Санкт–Петербургский государственный университет**

***Куклин Дмитрий Владимирович***

**Выпускная квалификационная работа**

# *Разработка нейро-нечёткой системы управления гоночным автомобилем в симуляторе TORCS*

Уровень образования: бакалавриат
Направление: 02.03.02 «Фундаментальная информатика и информационные технологии»
Основная образовательная программа: СВ.5003.2017 «Программирование и информационные технологии»

Научный руководитель: доцент, кафедра высшей математики, к.ф.-м.н. Басков Олег Владимирович

Рецензент: старший преподаватель, кафедра технологии программирования, Стученков Александр Борисович

Санкт-Петербург
2021 г.

# Содержание

# Введение

Для многих задач не существует строгих подходов, позволяющих получить точный результат за приемлемое время (к таким задачам относятся, например, NP-полные задачи). В таких случаях кажется естественным прибегнуть к использованию неточных и не обоснованных математически строго методов решения. Для формализации таких методов в 1994 году L. Zadeh ввёл понятие «мягких вычислений» [1]. Мягкие вычисления включают в себя, например, нечёткие системы, искусственные нейронные сети и генетические алгоритмы. Мягкие вычисления позволяют эффективно использовать человеческие знания, справляться с неопределенностью, неточностью и учиться адаптироваться к неизвестной или изменяющейся среде для повышения производительности [2].

Среди методов мягких вычислений за последние несколько лет наибольший прогресс был достигнут в области глубоких нейронных сетей. Однако нейронная сеть представляет собой «чёрный ящик», то есть интерпретация вывода решения, полученного нейронными сетями, является трудной задачей для человека. Тем не менее во многих прикладных областях интерпретируемость является критически важным качеством искусственного интеллекта. Так, несмотря на достижения глубоких нейронных сетей в задаче автономного управления автомобилем, согласно классификации систем автоматического вождения Общества автомобильных инженеров (Society of Automotive Engineers, SAE), на данный момент в полномасштабном производстве не находится ни одного автомобиля выше второго уровня автоматизации из пяти возможных [3]. Очевидно, что система автономного управления автомобилем обязана быть максимально надёжной и безопасной, однако гарантии безопасности не достичь без понимания внутреннего устройства системы.

Одним из преимуществ методов мягких вычислений является возможность их объединения в гибридные системы, которые позволяют как устранять недостатки, так и выделять сильные стороны каждой независимой системы. К гибридным системам относятся нейро-нечёткие системы, полученные объединением нечётких систем и искусственных нейронных

сетей. Нейро-нечёткие системы могут быть обучены как нейронные сети, и при этом сохранить близкие к человеческим способность представления знаний и способность к объяснению нечётких систем [4]. В результате нейронные сети становятся более интерпретируемыми, в то время как нечёткие системы становятся способными к обучению. Возможно также объединить нечёткие системы с генетическими алгоритмами для обучения, однако этот подход менее предпочтителен [5].

С целью подтвердить выдвинутое предположение о способностях нейро-нечётких систем к обучению и объяснению, а также с учётом необходимости систем управления транспортным средством оставаться интерпретируемыми, в рамках данной работы была разработана нейро-нечёткая система управления автомобилем в среде гоночного симулятора TORCS (The Open Racing Car Simulator).

## Постановка задачи

Конечной целью настоящей работы являлось создание контроллера гоночного автомобиля в среде симулятора TORCS с применением теории нейро-нечётких систем. В начале выполнения данной работы были поставлены следующие задачи:

1. Провести анализ существующих подходов к созданию гибридных систем, совмещающих системы нечёткого вывода с искусственными нейронными сетями.

2. Сформировать набор требований к предпочтительной архитектуре нейро-нечёткой системы и выбрать наиболее подходящую архитектуру согласно выставленным требованиям.

3. Собрать данные для обучения системы и обучить нейро-нечёткие компоненты системы управления автомобилем.

4. Реализовать контроллер в среде симулятора TORCS.

Наибольший приоритет в работе отдавался интерпретируемости итоговой системы, поэтому каждая вышеуказанная задача была решена с учётом наибольшей способности системы к объяснению.

## Обзор литературы

При тщательном поиске не нашлось ни одной работы, в которой бы применялись нейро-нечёткие сети при разработке контроллера автомобиля для симулятора TORCS.

Тем не менее симулятор TORCS является популярной средой для разработки и тестирования виртуальных водителей, основанных на теории мягких вычислений. Например, водители, основанные на нечёткой системе управления, рассматриваются в работах S. Mohammed и др. [6] и E. Onieva и др. [7]. Генетические алгоритмы для обучения использовались Y. Saez и др. [8], а также применялись для настройки параметров нечёткой системы в работах D. Perez и др. [9] и M. Salem и др. [10]. В работах S. Wang и др. [11] и A. Ganesh и др. [12] использовалось глубокое обучение с подкреплением. Приведённые статьи оказали большую помощь в понимании устройства системы управления гоночным автомобилем и возможностей приложения к системе управления методов мягких вычислений.

Системы управления реальным автомобилем, основанные на нечёткой логике, рассмотрены в работах J. Naranjo и др. [13] и E. Onieva и др. [14].

Математические основы нейро-нечётких систем описаны в книге H. Singh и Y. Lone [15]. Примеры использования различных архитектур нейро-нечётких систем приведены в соответствующих работах: в статьях D. Nauck и R. Kruse [16] и J.-S. Jang [17] для предсказания значений временного ряда Макей-Гласса были использованы архитектуры NEFCON (Neuro-Fuzzy Controller) и ANFIS (Adaptive Neuro-Fuzzy Inference System) соответственно, в статье H. Berenji и P. Khedkar [18] архитектура GARIC (Generalized Approximate Reasoning-based Intelligence Control) была применена к задаче балансировки шеста [19].

# Глава 1. Обзор нейро-нечётких систем

## 1.1 Системы нечёткого вывода

Множества, элементы которых имеют степень принадлежности к множеству на интервале $[0, 1]$, называются нечёткими и были впервые введены L. Zadeh в 1965 году [20]. Более формально, пусть $X$ — множество объектов и $x$ — элемент $X$. Тогда нечёткое множество $A$ в $X$ описывается функцией принадлежности $f_A(x)$, которая сопоставляет каждому элементу $X$ вещественное число в интервале $[0, 1] \subset \mathbb{R}$. Значение $f_A(x)$ в точке $x$ описывает степень принадлежности $x$ к $A$: чем ближе значение $f_A(x)$ к единице, тем она выше. Если значение степени принадлежности равно 0, то объект не принадлежит данному нечёткому множеству, а если равно 1, то объект полностью принадлежит множеству. Наиболее часто используются треугольные и трапециевидные функции принадлежности, а также функция Гаусса (см. Уравнение 1)

$$f_A(x) = e^{-\left(\frac{x-b}{a}\right)^2}, \tag{1}$$

где $a$ и $b$ — произвольные параметры и функция «обобщённый колокол» (см. Уравнение 2)

$$f_A(x) = \frac{1}{1 + \left|\frac{x-c}{a}\right|^{2b}}, \tag{2}$$

где $a$, $b$ и $c$ — произвольные параметры. В нейро-нечётких системах, как правило, используются последние две функции, так как упрощаются вычисления в алгоритме обратного распространения ошибки.

На основе нечётких множеств было введено понятие нечёткого вывода [21]. Процесс нечёткого вывода состоит из следующих шагов:

1. Фаззификация входных значений.
2. Применение правил нечёткого вывода.
3. Агрегация выходных значений правил вывода.
4. Дефаззификация.

Фаззификация — это процесс преобразования чёткого входного значения $x \in \mathbb{R}$ в нечёткое значение $f_A(x) \in A$ для каждой функции принадлежности $A$ из базы знаний нечёткой системы. Преимущество использования нечётких множеств состоит в том, что функциям принадлежности и нечётким множествам даются характеристики на естественном языке, поэтому после фаззификации процесс вывода становится похожим на процесс человеческого размышления. Пример фаззификации значения температуры воды приведён на Рисунке 1.

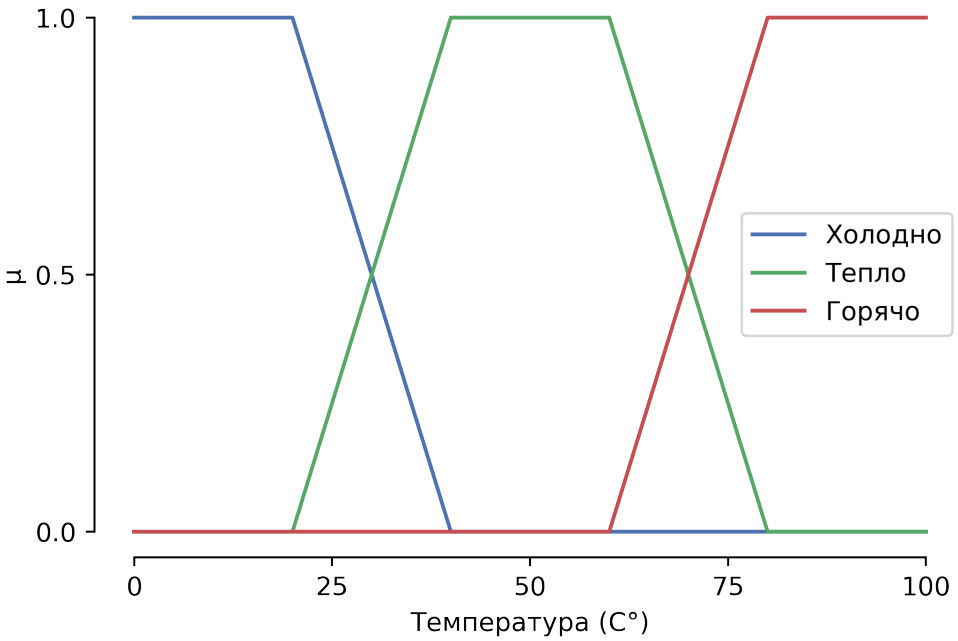



**Рис. 1:** Фаззификация значения температуры воды

Как и в классической логике, modus ponens и modus tollens являются основными правилами нечёткого вывода. Правило modus ponens имеет вид

$$\frac{x \text{ ЕСТЬ } A \qquad x \text{ ЕСТЬ } A \implies y \text{ ЕСТЬ } B}{y \text{ ЕСТЬ } B}$$

Правила нечёткого вывода принято записывать в виде

$$R_i: \text{ ЕСЛИ } x_1 \text{ ЕСТЬ } A_{i1} \text{ И } \ldots \text{ И } x_m \text{ ЕСТЬ } A_{im}, \text{ ТО } y \text{ ЕСТЬ } B_i$$
$$\forall i \in \{1, \ldots n\},$$

где $n$ — число правил, $m$ — число переменных нечёткой системы. В нечётком правиле предпосылка «$x$ ЕСТЬ $A$» и следствие «$y$ ЕСТЬ $B$» могут быть истинными не полностью, а в какой-либо степени, так как $A$ и $B$ являются нечёткими множествами. Для вычисления результатов операций конъюнкции и дизъюнкции могут быть использованы любые функции, являющиеся $T$-нормой и $T$-конормой соответственно.

Дефаззификация — это процесс обратный фаззификации, то есть преобразование нечёткого выходного значения в чёткое, вещественное число. В данной работе используется система вывода типа Сугено, в которой значение вывода каждого правила $R_i$ принимается за силу $w_i \in [0, 1]$ этого правила. В системе вывода типа Сугено вывод системы считается как взвешенное среднее выводов всех правил (см. Уравнение 3)

$$\frac{\sum_{i=1}^{n} w_i z_i}{\sum_{i=1}^{n} w_i}, \tag{3}$$

где $z_i$ — линейная функция от входных переменных $z_i = f(x_1, \ldots, x_m)$. На практике наиболее часто в качестве $z_i$ используются константные значения.

Тем не менее более интерпретируемые результаты получаются с использованием вывода типа Мамдани. В таком типе вывода выход каждого правила представляет собой нечёткое множество, полученное из выходной функции принадлежности с применением метода импликации (например, $\min$). Далее выходные нечёткие множества объединяются в единое нечёткое множество с использованием метода агрегирования (например, $\max$). Для вычисления конечного выходного значения объединённое выходное нечёткое множество дефаззифицируется с помощью, например, метода центроида (см. Уравнение 4)

$$\frac{\sum_{i=1}^{n} \mu(x_i) x_i}{\sum_{i=1}^{n} \mu(x_i)}. \tag{4}$$

Очевидно, такой тип вывода является вычислительно более затратным в сравнении с выводом типа Сугено.

## 1.2 Искусственные нейронные сети

Искусственный нейрон — это программная конструкция, стремящаяся имитировать поведение биологического нейрона в мозге [22]. Искусственный нейрон $j$, получающий входные данные $p_j(t)$ от предшествующих нейронов, содержит следующие компоненты (см. Рисунок 2):

1. Активация $a_j(t)$, состояние нейрона в момент времени $t$.
2. Пороговое значение $\theta_j$, изменяемое в момент обучения.
3. Функция активации $f$, которая вычисляет новую активацию в момент времени $t+1$ как $a_j(t+1) = f(a_j(t), p_j(t), \theta_j)$.
4. Выходная функция $o_j(t) = f_{\text{out}}(a_j(t))$.

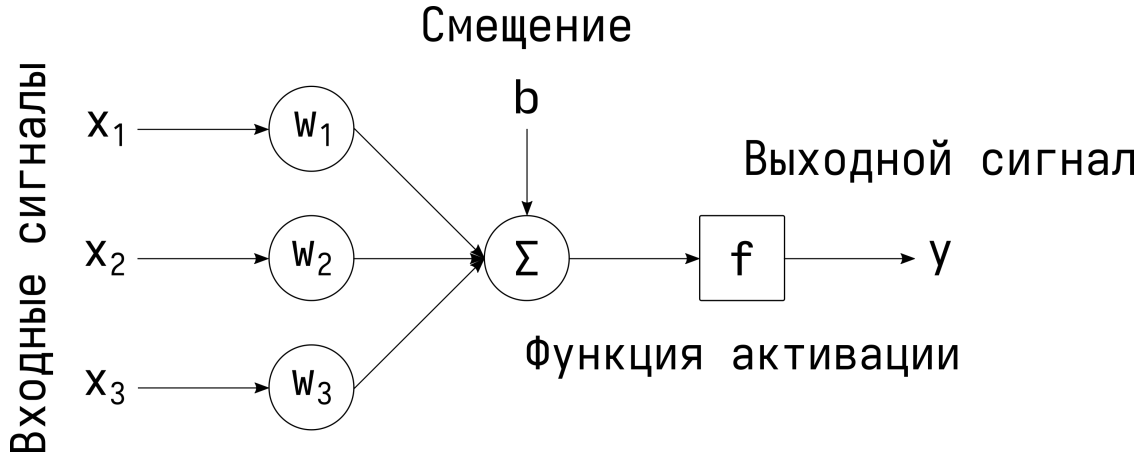



**Рис. 2:** Вид нейрона с тремя входами

Исключениями являются входные нейроны, которые не имеют предшествующих нейронов. Функция распространения, вычисляющая вход $p_j(t)$ нейрона $j$ по значениям выходных функций $o_i(t)$, как правило, имеет вид $p_j(t) = \sum_i o_i(t) w_{ij} + w_{0j}$, где $w_{ij}$ — коэффициент, с которым выходное значение нейрона $i$ входит в нейрон $j$, а $w_{0j}$ — смещение. Функция активации может принимать любой вид, например, в качестве функции активации зачастую используются сигмоида (см. Уравнение 5).

$$a_j(x) = \sigma(x) = \frac{1}{1 + e^{-x}}, \tag{5}$$

а также разновидности функции «линейного выпрямителя» (см. Уравнение 6)

$$a_j(x) = \max\{0, x\}. \tag{6}$$

Искусственные нейронные сети можно представить в виде ориентированного графа. Нейрон представляется в виде узла $i$, а направленное ребро $e = (i, j)$ указывает, что один из входов нейрона $j$ является выходом нейрона $i$. Ребро $(i, j)$ помечается соответствующим весом $w_{ij}$.

Наборы узлов, которые вычисляются вместе, известны как слои. Первый слой называется входным слоем, последний слой называется выходным слоем, остальные слои считаются промежуточными. Далее в работе при определении структуры нейронных сетей учитываются лишь промежуточные слои.

Искусственные нейронные сети применяются для оптимизации и обучения функций. Процесс обучения заключается в предоставлении нейронной сети набора пар $(\vec{x_i}, \vec{y_i})$ и обучении модели аппроксимации функции $f$ так, что $f(\vec{x_i}) = \vec{y_i}$ для всех пар из заданного набора. Таким образом, если размерность вектора $\vec{x}$ равна $n$, а размерность вектора $\vec{y}$ равна $m$, то итоговый граф нейронной сети будет содержать $n$ входных узлов, произвольное число внутренних узлов и $m$ выходных узлов.

Значения векторов весов $\vec{w_i}$ и смещений отдельных узлов известны как параметры нейронной сети $\theta$. Для обучения функции $f_\theta(\vec{x})$, зависящей от параметров $\theta$, требуется изменить эти смещения и весовые векторы так, чтобы нейронная сеть вычисляла известные значения функции. То есть, с учётом ряда пар ввода-вывода $X = \left\{(\vec{x_1}, \vec{y_1}), \ldots, (\vec{x_n}, \vec{y_n})\right\}$, требуется изменить весовые векторы и смещения таким образом, чтобы $\forall i\ f_\theta(\vec{x_i}) \approx \vec{y_i}$. Подбираются параметры, минимизирующие заданную ошибку $E(X, \theta)$, например, среднеквадратичную ошибку. Для подбора параметров используется метод градиентного спуска, а для нахождения оптимальных параметров узлов всей сети используется алгоритм обратного распространения ошибки, согласно которому параметры на шаге $t$ с заданным параметром $\alpha$

обновляются по формуле

$$\theta^t = \theta^{t-1} - \alpha \frac{\partial E(X, \theta^{t-1})}{\partial \theta}.$$

## 1.3 Обзор архитектур нейро-нечётких систем

Согласно определению, данному D. Nauck [23], нейро-нечёткие системы обязаны удовлетворять следующим требованиям:

1. Нейро-нечёткая система — это нечёткая система, обученная алгоритмом, выведенным из теории нейронных сетей.

2. Нейро-нечёткая система может быть представлена в виде нейронной сети с прямой связью с одним и более слоями. В качестве функций активации нейроны нечётких нейронных сетей обязаны использовать функции, являющиеся $T$-нормой и $T$-конормой.

3. Нейро-нечёткая система как до, так и после обучения может быть представлена в виде системы нечёткого вывода.

4. Обучающая процедура нейро-нечёткой системы принимает во внимание семантические свойства лежащей в основе нечёткой системы. Таким образом накладываются ограничения на возможные модификации параметров системы.

5. Нейро-нечёткая система аппроксимирует $n$-мерную функцию, частично определённую обучающими данными.

Таким образом, при условии ограничений, накладываемых приведённым определением, количество рассматриваемых архитектур нейро-нечётких систем было значительно сокращено. В итоге для решения поставленной задачи наиболее подходящими оказались следующие архитектуры:

1. Адаптивная нейро-нечёткая система вывода (Adaptive Neuro-Fuzzy Inference System, ANFIS).

2. Нейро-нечёткий контроллер (Neuro-Fuzzy Controller, NEFCON).

3. Обобщенный приближенный интеллектуальный контроль на основе рассуждений (Generalized Approximate Reasoning-based Intelligence Control, GARIC).

4. Нечёткая адаптивная обучаемая сеть управления (Fuzzy Adaptive Learning Control Network, FALCON)

## 1.4 Выбор архитектуры

Краткий обзор наиболее популярных архитектур гибридных нейро-нечётких сетей приведён в статье A. Ajith [24]. Более подробный разбор принципов работы архитектур и областей их применения представлен в статье K. Kis и Zs. Viharos [25]. Основные свойства каждой архитектуры приведены в Таблице 1.

| Архитектура | Тип вывода | Изменяемая структура |
|---|---|---|
| ANFIS | Сугено | Нет |
| NEFCON | Мамдани | Да |
| GARIC | Особый | Нет |
| FALCON | Мамдани | Да |

**Таблица 1:** Свойства архитектур нейро-нечётких систем

Как было обозначено ранее, основная задача данной работы состоит в создании интерпретируемой и в то же время оптимальной нечёткой системы. Тем не менее определённые в Подразделе 3.1 правила достаточно просты для интерпретации, поэтому ни дополнения, ни сокращения базы правил вероятно не потребуются. Кажется естественным использовать систему вывода типа Сугено вместо системы вывода типа Мамдани, так как несмотря на то, что последняя предоставляет в общем более интерпретируемые результаты, используемые в работе константные значения выходных переменных просты для понимания и в то же время приводят к лучшим результатам.

Дополнительно стоит обратить внимание на доступность программных средств для реализации приведённых ранее архитектур. Так, помимо

базовых реализаций нечётких нейросетевых архитектур от авторов соответствующих статей, не было найдено ни одной современной библиотеки, реализующей указанные выше архитектуры, за исключением архитектуры ANFIS. В том числе по этой причине для реализации нечёткого контроллера было решено выбрать адаптивную нейро-нечёткую систему вывода.

## 1.5 Адаптивная нейро-нечёткая система вывода

Адаптивная нейро-нечёткая система вывода, или ANFIS, была предложена J.-S. Jang в 1993 году [17]. ANFIS представляет собой базу знаний нечёткой системы в виде нейронной сети (см. Рисунок 3). В данной архи-

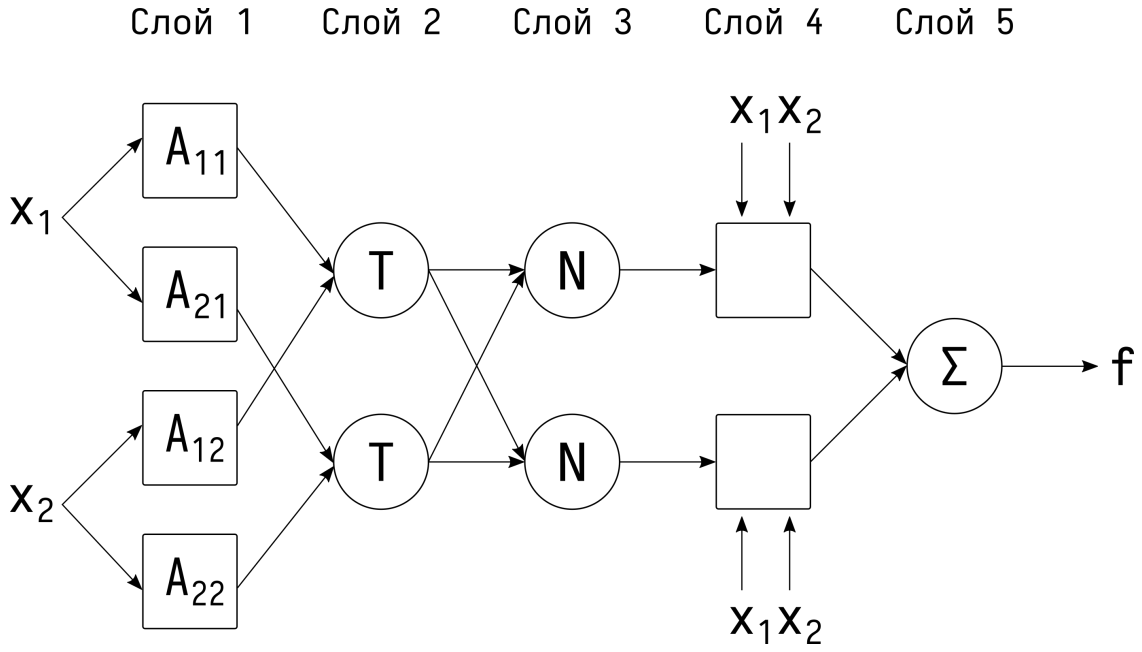



**Рис. 3:** ANFIS с двумя входными параметрами и двумя правилами

тектуре правила нечёткого вывода принимают вид:

$$\begin{gathered}
\text{ЕСЛИ } x_1 \text{ ЕСТЬ } A_{11} \text{ И } \ldots \text{ И } x_m \text{ ЕСТЬ } A_{1m}, \text{ ТО } y \text{ ЕСТЬ } f_1(x_1, \ldots, x_m), \\
\text{ЕСЛИ } x_1 \text{ ЕСТЬ } A_{21} \text{ И } \ldots \text{ И } x_m \text{ ЕСТЬ } A_{2m}, \text{ ТО } y \text{ ЕСТЬ } f_2(x_1, \ldots, x_m), \\
\ldots \\
\text{ЕСЛИ } x_1 \text{ ЕСТЬ } A_{n1} \text{ И } \ldots \text{ И } x_m \text{ ЕСТЬ } A_{nm}, \text{ ТО } y \text{ ЕСТЬ } f_n(x_1, \ldots, x_m).
\end{gathered}$$

В приведённых выше правилах $A_{ij}$ — нечёткое множество, $x_i$ — входная переменная, $y$ — выходная переменная и $f_i(x_1, \ldots, x_m)$ — произвольная функция от входных переменных.

Архитектура ANFIS в представлении нейронной сети состоит из пяти слоёв.

В первом слое вычисляются значения функций принадлежности для каждой входной переменной $x_i$ на каждом нечётком множестве $A_{ij}$. Как правило, в ANFIS используются либо функция Гаусса (с двумя параметрами), либо «обобщённый колокол» (с тремя параметрами). Параметры функций принадлежности в этом слое подбираются при помощи алгоритма обратного распространения ошибки (всего $k \times m \times n$ параметров, где $k$ — число параметров функции принадлежности).

Во втором слое вычисляются степени активации $w_i$ каждого правила. Степень активации вычисляется с использованием заранее заданной функции $T$-нормы, например, $T(A_{i1}, A_{i2}, \ldots, A_{im}) = A_{i1} \times A_{i2} \times \ldots \times A_{im}$.

В третьем слое степени активации нормализуются (см. Уравнение 7)

$$\overline{w_i} = \frac{w_i}{\sum_j w_j}. \tag{7}$$

В четвёртом слое вычисляются итоговые выходные значения каждого правила $y_i = \overline{w_i} f_i(x_1, \ldots, x_m)$. Если выходная функция является константой, то $y_i = \overline{w_i} c_i$ и оптимизируется один параметр $c_i$, если функция линейна по входным переменным, то оптимизируются $(m + 1) \times n$ параметров. Параметры в этом слое подбираются с использованием метода наименьших квадратов.

В последнем пятом слое вычисляется общий вывод системы как взвешенное среднее выходных значений каждого правила $y = \sum_i \overline{w_i} f_i(x_1, \ldots, x_m)$.

Таким образом, при обучении системы оптимизируются как параметры функций принадлежности, так и параметры выходных функций каждого правила.

# Глава 2. Программная реализация контроллера

## 2.1 TORCS

TORCS (The Open Racing Car Simulator) [26] — это гоночный симулятор с открытым исходным кодом, предоставляющий удобный функционал для внедрения и испытания контроллеров гоночных автомобилей (см. Рисунок 4). Официальная версия симулятора позволяет описывать

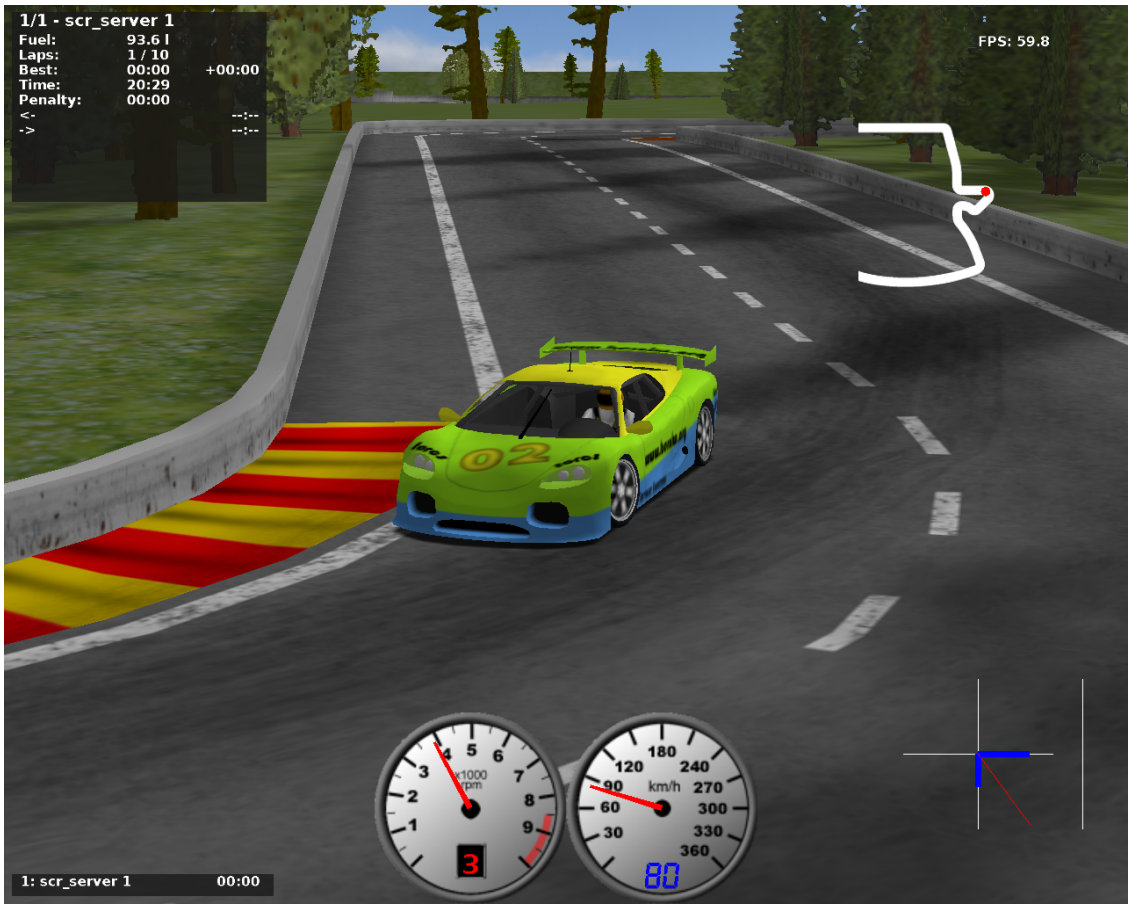



**Рис. 4:** Снимок экрана TORCS

логику контроллера на языке программирования C++ и далее загружать скомпилированную программу водителя в качестве отдельного модуля. Однако такой подход не используется в соревнованиях последних лет. Для проведения соревнований была разработана версия TORCS, TORCS SCR (Simulated Car Racing Championship) [27], которая представляет собой клиент-серверное приложение: сервер TORCS передаёт каждому клиенту-участнику заезда информацию о состоянии автомобиля и окружения и в ответ прини-

мает команды по управлению автомобилем. Стоит отметить, что процесс обмена информацией происходит с периодичностью в 20 миллисекунд, поэтому контроллер обязан быть вычислительно эффективным.

Информация о состоянии автомобиля и окружения в версии TORCS SCR ограничена в сравнении с оригинальной версией и представлена 79 сенсорами (см. Приложение А.1). В начале каждого заезда клиент указывает 19 значений углов, на которых будут установлены сенсоры расстояний между автомобилем и краем трассы (см. Рисунок 5).

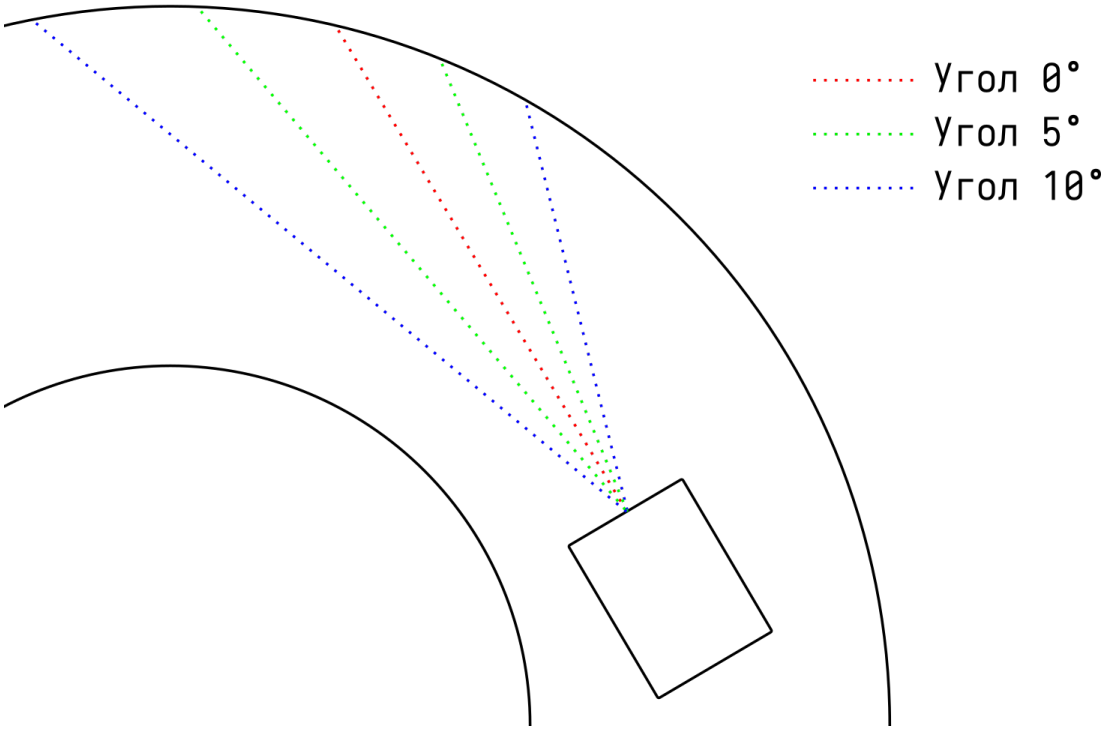



**Рис. 5:** Расположение сенсоров `track` при разных углах

В каждом цикле взаимодействия клиента с сервером клиент обязан отправлять информацию о следующих семи командах: сила нажатия на педаль газа, сила нажатия на педаль тормоза, сила нажатия на педаль сцепления, значение передачи, угол поворота рулевого колеса, направление взгляда и запрос о перезапуске сервера (см. Приложение А.2).

## 2.2 Библиотека Čapek

В отрытом доступе не оказалось готовых к использованию библиотек на языке Python, которые бы позволили реализовать логику контроллера,

и при этом взять на себя задачу по взаимодействию с сервером. Такая библиотека была написана и выложена в открытый доступ автором данной работы [28]. Библиотека Čapek позволяет создать класс-наследник водителя и переопределить необходимые методы, такие как действие на каждом шаге цикла обмена информацией, действие при окончании или перезапуске заезда. Пример программы, использующей библиотеку, приведен в Листинге 1.

```
from capek import Driver, Client
from control import SpeedControl, SteeringControl

class MyDriver(Driver):
    speed_control = SpeedControl(min=20, max=320)
    steering_control = SteeringControl()

    def drive(self, state: List, control: List) -> List:
        control.accel, control.brake = speed_control.tick(state)
        control.steering = steering_control.tick(state)

        return control

client = Client(verbosity=1, port=3002)
client.run(driver=MyDriver)
```

**Листинг 1:** Пример клиентского кода

## 2.3 Библиотека FuzzyLite

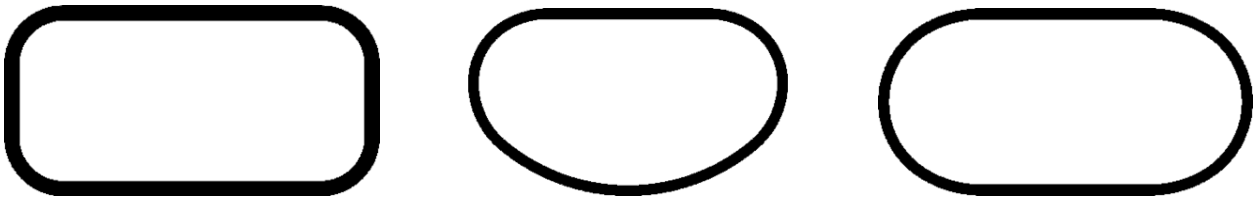
Нечёткие модули разработанного контроллера используют библиотеку FuzzyLite для языка Python [29]. Данная библиотека предоставляет возможность описания нечёткой системы вывода в текстовом виде с указанием функций принадлежности, их параметров, и правил вывода. Библиотека работает с системой вывода типа Сугено и включает функции принадлежности Гаусса и «обобщённый колокол».

## 2.4 Сбор данных

Для обучения модели нейро-нечёткой системы требовалось собрать данные заездов других контроллеров. Данные собирались с виртуальных гонщиков, участвовавших в соревнованиях, проводившихся на конферен-

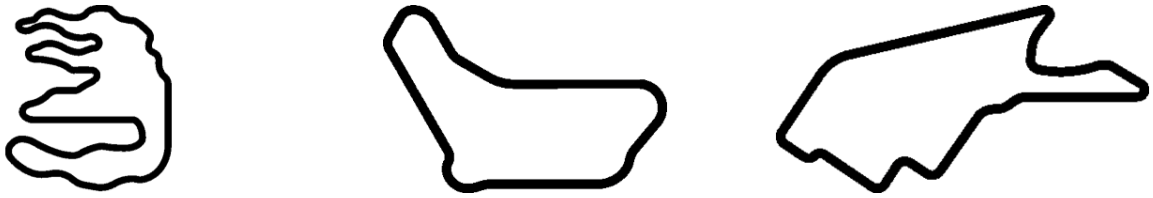
ции GECKO в 2012 и 2013 годах. Код участников находится в открытом доступе и доступен для скачивания [30].

В базовой версии TORCS имеется 37 трасс: 8 кольцевых трасс (см. Рисунок 6), 20 стандартных гоночных трасс (см. Рисунок 7) и 9 грунтовых трасс (см. Рисунок 8).



**(a)** Трасса Speedway A **(b)** Трасса Speedway G **(c)** Трасса Speedway C

**Рис. 6:** Кольцевые трассы TORCS



**(a)** Трасса Alpine **(b)** Трасса CG Track 2 **(c)** Трасса Street 1

**Рис. 7:** Стандартные гоночные трассы TORCS

**(a)** Трасса Dirt 2 **(b)** Трасса Dirt 4 **(c)** Трасса Mixed 2

**Рис. 8:** Грунтовые трассы TORCS

Контроллерам гоночных автомобилей были поставлены следующие требования:

1. Виртуальный гонщик обязан дойти до финиша на каждой трассе.
2. Виртуальный гонщик обязан пройти все стандартные и кольцевые трассы без получения урона.
3. Виртуальный гонщик обязан пройти все кольцевые трассы без съезда с области трассы ($|\texttt{trackPos}| \leq 1$).

4. Максимальная скорость автомобиля по результатам заезда должна превышать 150 километров в час.

По результатам испытаний выставленным требованиям соответствовали 2 гоночных робота: Autopia и GRNDriver.

Помимо популярности версии TORCS SCR в сравнении со стандартной версией TORCS, выбор в пользу TORCS SCR был обоснован относительной простотой процесса сбора данных виртуальных гонщиков. Так как TORCS SCR представляет собой клиент-серверное приложение, для сбора данных не потребовалось изучать и модифицировать код на разных языках программирования. Для процедуры сбора данных было разработано промежуточное программное обеспечение (ПО). Разработанное ПО принимает информацию о состоянии автомобиля от сервера и передаёт полученную информацию клиенту. Аналогично, ПО принимает информацию об управлении от клиента и передаёт полученную информацию серверу. Данные о состоянии автомобиля и управлении, полученные в описанном процессе, сохраняются как цельная строка таблицы данных. По окончании заезда данные обрабатываются и сохраняются в CSV-файле. Архитектура разработанного ПО продемонстрирована на Рисунке 9.

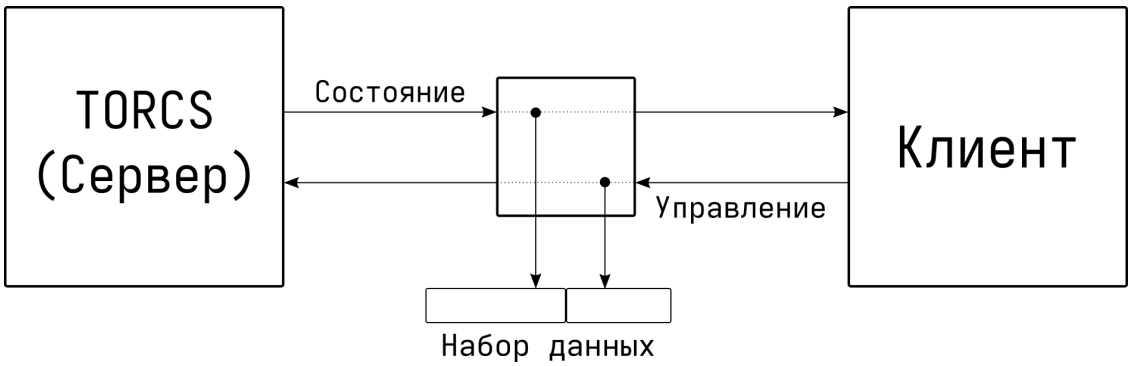



**Рис. 9:** Промежуточное ПО для сбора данных

Каждый гоночный робот, соответствующий заданным выше требованиям, преодолел 10 последовательных кругов (заезд) на каждой стандартной и кольцевой трассе (всего 28 трасс). Было принято решение не использовать данные с грунтовых трасс вследствие наличия большого количества спусков и подъемов, при том что итоговый контроллер не использует значения скорости по оси $z$ `speedZ` и высоты подъема `z`. Контроллер Autopia,

несмотря на лучшие результаты в сравнении с контроллером GRNDriver, иногда сходил с трассы в заездах на стандартных трассах (см. Приложение А.4), поэтому обучающая и валидационная выборки формировались с использованием библиотеки Pandas [31] следующим образом:

1. В обучающую выборку были включены данные трёх кругов с лучшим временем для гоночного робота GRNDriver для каждой трассы.

2. Для гоночного робота Autopia были выбраны данные круга с лучшим временем для заездов, в которых контроллер оставался в пределах трассы.

В итоге данные были поделены приблизительно в отношении 5 к 1 на обучающую и валидационную выборки соответственно.

## 2.5 Библиотека ANFIS PyTorch

Для обучения нечётких модулей контроллера использовалась реализация ANFIS [32] на основе фреймворка PyTorch [33]. Библиотека предоставляет удобный интерфейс для задания и вывода параметров настроенной системы, что позволило без особых проблем интегрировать её с библиотеками Čapek и FuzzyLite, а также с обработанными с использованием Pandas данными.

Дополнительно в состав библиотеки входили примеры оптимизации, в том числе из оригинальной статьи J.-S. Jang [17], что способствовало быстрому освоению данной библиотеки.

# Глава 3. Архитектура контроллера

В данной работе задача управления поворотом рулевого колеса и задача повышения и снижения скорости автомобиля были разделены. Несмотря на то, что угол поворота автомобиля и набранная автомобилем скорость влияют друг на друга, в гоночном симуляторе взаимное влияние не существенно.

Для определения значений силы нажатия на педаль газа и тормоза сравнивалась разность значений желаемой скорости и текущей скорости автомобиля по аналогии с модульной системой из работы E. Onieva и др. [7].

## 3.1 Нечёткие модули контроллера

### 3.1.1 Модуль желаемой скорости

Для определения желаемой скорости (заданной выходной переменной `DesiredSpeed`) было решено выбрать 4 переменные:

1. `Angle` — абсолютное значение угла отклонения оси автомобиля от оси центра трассы $|\texttt{angle}|$.
2. `Position` — абсолютное значение отклонения от центра трассы $|\texttt{trackPos}|$.
3. `Front` — значение `track` на углу $0°$ (см. Рисунок 5).
4. `Corner` — максимум из значений `track` на углах $-10°$ и $10°$ (см. Рисунок 5).

Для каждой указанной переменной эмпирически, с оглядкой на полученные данные, были определены три функции принадлежности вида «обобщённый колокол» (см. Рисунки 10–12).

Для каждой возможной комбинации лингвистических переменных были сгенерированы ($3^4 = 81$) правила нечёткого вывода, как в Уравнении 8. Опытным путём было установлено, что система типа Сугено с более чем 120 входными правилами, либо система типа Мамдани с более чем 75 входными правилами не успевают вычислять выходные значения за один

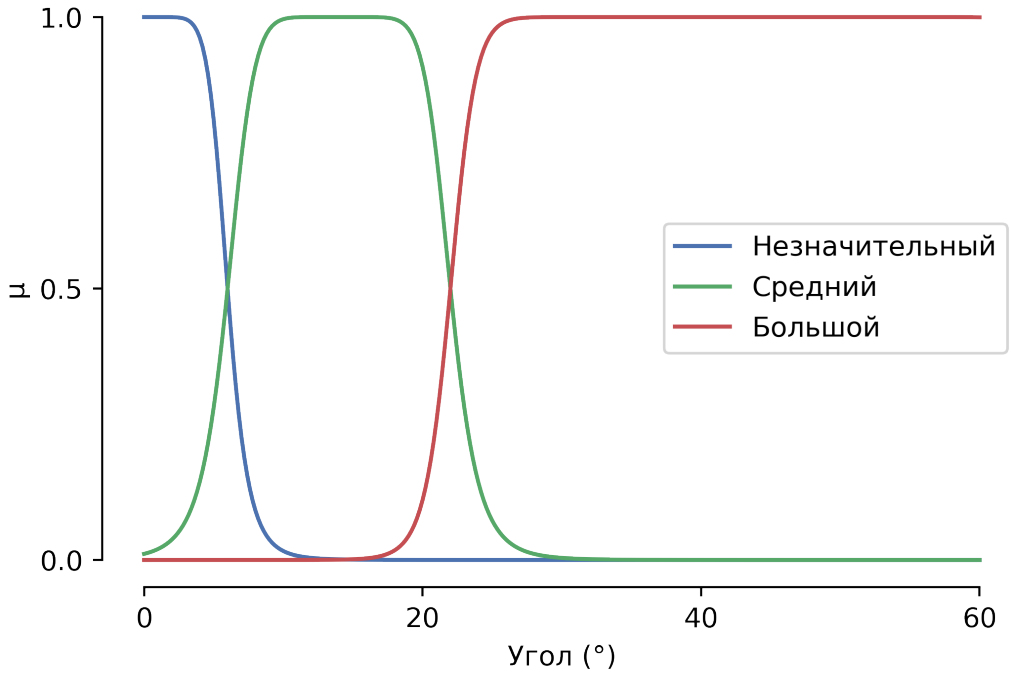



**Рис. 10:** Функции принадлежности переменной `Angle`

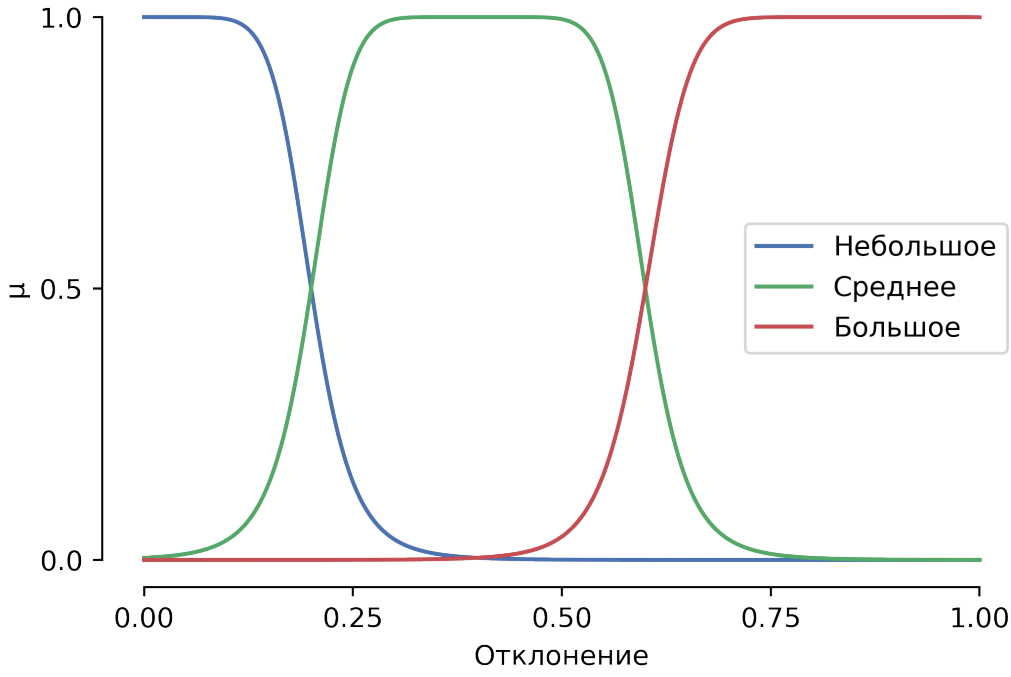



**Рис. 11:** Функции принадлежности переменной `Position`

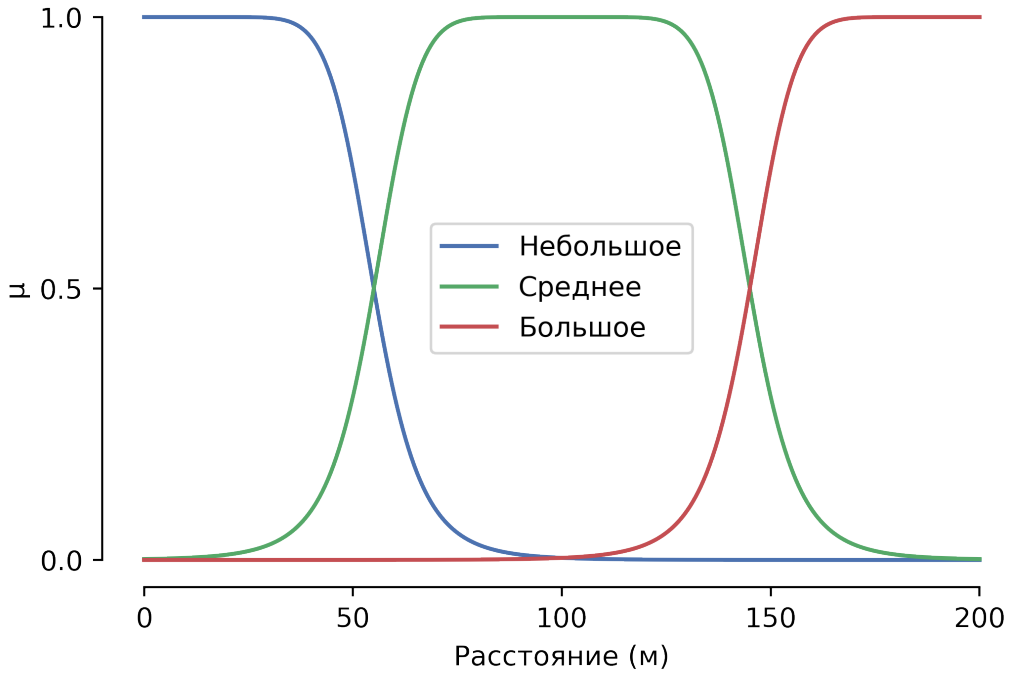



**Рис. 12:** Функции принадлежности переменных `Front` и `Corner`

цикл обмена данными. Таким образом было выбрано оптимальное число входных переменных и функций принадлежности.

$$\text{ЕСЛИ} \left\{ \begin{array}{lll} \texttt{Angle} & \text{ЕСТЬ} & A_i \\ \texttt{Position} & \text{ЕСТЬ} & B_i \\ \texttt{Front} & \text{ЕСТЬ} & C_i \\ \texttt{Corner} & \text{ЕСТЬ} & D_i \end{array} \right., \text{ ТО DesiredSpeed ЕСТЬ } E_i \qquad (8)$$

Одним из ограничений архитектуры ANFIS является необходимость создания уникальной выходной переменной для каждого заданного правила. В итоге перед обучением была задана 81 константная выходная переменная, которым были присвоены равноудалённые значения из области определения `DesiredSpeed` $y \in [0, 320]$.

Описанная система нечёткого вывода была обучена на собранных данных (см. Подраздел 2.4) с использованием библиотеки ANFIS PyTorch (см. Подраздел 2.5). При обучении в качестве переменной `DesiredSpeed` была взята переменная `speedX` из таблицы данных. Так как функции при-

надлежности «обобщённый колокол» содержат 3 параметра, оптимизировались 36 параметров во втором слое и 81 параметр для каждой выходной переменной в четвёртом слое системы.

В результате обучения параметры функций принадлежности практически не изменились (максимальное изменение параметра $c$, центра колокола, составило 8%), однако значения выходных констант вышли далеко за обозначенные пределы в $[0, 320]$ километров в час. Так, максимальное абсолютное выходное значение составляет $\sim$4594 километров в час. Тем не менее в процессе вывода чёткого значения веса усредняются и получаются приемлемые значения скорости, не выходящие за заданную область определения. Модель сошлась за 24 полных цикла обучения, итоговая среднеквадратическая ошибка системы на обучающей выборке составила 24.6, на валидационной выборке — 28.5.

Получившиеся значения выходных переменных удобны для интерпретации. Например, в Уравнении 9 вероятно описывается ситуация, в которой автомобиль поворачивает, однако находится по центру трассы. В таком случае разумно развить достаточно высокую скорость.

$$\text{ЕСЛИ} \left\{ \begin{array}{lll} \texttt{Angle} \text{ (Угол)} & \text{ЕСТЬ} & \text{Незначительный} \\ \texttt{Position} \text{ (Отклонение)} & \text{ЕСТЬ} & \text{Небольшое} \\ \texttt{Front} \text{ (Расстояние, 0°)} & \text{ЕСТЬ} & \text{Небольшое} \\ \texttt{Corner} \text{ (Расстояние, 10°)} & \text{ЕСТЬ} & \text{Среднее} \end{array} \right\}, \quad (9)$$

$$\text{ТО DesiredSpeed ЕСТЬ 230.9}$$

Однако, в Уравнении 10 отклонение от центра трассы значительное, поэтому скорость логично снизить.

$$\text{ЕСЛИ} \left\{ \begin{array}{lll} \texttt{Angle} \text{ (Угол)} & \text{ЕСТЬ} & \text{Незначительный} \\ \texttt{Position} \text{ (Отклонение)} & \text{ЕСТЬ} & \text{Большое} \\ \texttt{Front} \text{ (Расстояние, 0°)} & \text{ЕСТЬ} & \text{Небольшое} \\ \texttt{Corner} \text{ (Расстояние, 10°)} & \text{ЕСТЬ} & \text{Среднее} \end{array} \right\}, \quad (10)$$

$$\text{ТО DesiredSpeed ЕСТЬ 125.8}$$

К сожалению, полученный контроллер не смог пройти все стандартные трассы TORCS, так как скорость при вхождении в поворот иногда превышала оптимальную, что объясняется ошибками контроллера Autopia. Для решения обозначенной проблемы при резком повороте значение желаемой скорости снижалось на $60 - \texttt{Front}$, если $\texttt{Front} < 20$ и $\texttt{Front} < \frac{\texttt{Corner}}{2}$.

### 3.1.2 Модуль управления рулевым колесом

Для определения поворота рулевого колеса (заданного выходной переменной `Steering`) были выбраны те же переменные, что и для переменной `DesiredSpeed`. Аналогично были определены функции принадлежности и правила нечёткого вывода (см. Уравнение 11). Нечёткая система использовалась для определения абсолютного значения поворота, поэтому константным выходным переменным были заданы равноудалённые значения из области $[0, 1]$.

$$\text{ЕСЛИ} \left\{ \begin{array}{lll} \texttt{Angle} & \text{ЕСТЬ} & A_i \\ \texttt{Position} & \text{ЕСТЬ} & B_i \\ \texttt{Front} & \text{ЕСТЬ} & C_i \\ \texttt{Corner} & \text{ЕСТЬ} & D_i \end{array} \right\}, \text{ ТО } \texttt{Steering} \text{ ЕСТЬ } E_i \qquad (11)$$

В результате обучения, как и в случае `DesiredSpeed`, значения выходных переменных превышали пределы области (максимальное абсолютное выходное значение $\sim$88).

Для определения направления поворота использовались значения угла `track` при $10°$ и $-10°$ градусах. Так, если расстояние при угле в $10°$ меньше расстояния при угле в $-10°$, то поворот левый. Для правого поворота аналогично.

Во избежание излишнего маневрирования на прямых участках трассы, если абсолютная разность расстояний на углах $10°$ и $-10°$ градусах меньше 10 метров, то значение поворота рулевого колеса не вычисляется и принимается равным 0.

Аналогично проблеме снижения скорости при резком повороте в Подразделе 3.1.1, был введён дополнительный контроль за выравниванием ав-

томобиля по центру трассы. Так, если `Position` $> 0.75$, то поворот рулевого колеса изменяется на $0.75 -$ `Position`. Соответственно, если `Position` $< -0.75$, то поворот изменяется на $-0.75 -$ `Position`.

## 3.2 Остальные модули контроллера

### 3.2.1 Модуль переключения передачи

Модуль переключения передачи указывает водителю нажать на педаль сцепления и понизить передачу, если значение числа оборотов в минуту снизилось до заданного. Аналогично для повышения передачи. В Таблице 2 приведены значения числа оборотов в минуту, при которых производится переключение передачи.

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Увеличение | 8000 | 8000 | 8000 | 8000 | 8000 | |
| Понижение | | 2500 | 3000 | 3000 | 3500 | 3500 |

**Таблица 2:** Число оборотов в минуту, требуемых для изменения передачи

Дополнительно, поскольку TORCS является детально проработанным симулятором, во избежание застревания переключение передачи не производится, если с момента последнего изменения передачи произошло менее 7 циклов обмена данными.

### 3.2.2 Модуль поддержания оптимальной скорости

Модуль поддержания оптимальной скорости контролирует значения нажатия на педаль газа и тормоза в зависимости от разницы между текущей скоростью автомобиля и желаемой скоростью, полученной в соответствующем модуле. На вход модуля подаются значения `speedX` и `DesiredSpeed`. Для вычисления силы нажатия на каждую из педалей используется сигмоидальная кривая (см. Уравнение 12)

$$S = \frac{2}{1 + e^{\Delta}}, \text{ где } \Delta = \texttt{speedX} - \texttt{DesiredSpeed}. \tag{12}$$

Если $S \geq 1$, то сила нажатия на педаль газа равна $S - 1$, а сила нажатия на педаль тормоза равна нулю. И аналогично если $S < 1$, то сила нажатия на педаль тормоза равна $1 - S$, а сила нажатия на педаль газа равна нулю. Таким образом, в каждый момент времени нажата лишь одна педаль.

Во избежание проскальзывания и последующих заносов при резком торможении был введён контроль за торможением наподобие АБС. Так, если текущая скорость автомобиля выше заданной минимальной скорости АБС, среднее проскальзывание колёс (см. Уравнение 13)

$$p = \frac{R \sum_{i=1}^{4} \texttt{wheel\_spin\_velocity}_i}{4 \cdot \texttt{speedX}} \tag{13}$$

велико и среднее значение проскальзывания выше заданного минимума, то сила нажатия на педаль газа снижается пропорционально значению проскальзывания $\texttt{brake} = p \cdot \texttt{brake}$. В Уравнении 13 $R$ — радиус колёс автомобиля, `wheel_spin_velocity` — скорость вращения колеса и `speedX` — текущая скорость автомобиля.

## Результаты

Для сравнительной оценки качества обученного контроллера была создана нечёткая система, модули которой используют начальные функции принадлежности из Подраздела 3.1. Выходному значению каждого правила для `DesiredSpeed` была присвоена одна из 6 констант $\mathrm{DS}_1, \ldots, \mathrm{DS}_6$, а выводу каждого правила для `Steering` — одна из 8 констант $\mathrm{S}_1, \ldots, \mathrm{S}_8$, согласно интуиции и опыту ранее проведённых заездов. Правила нечёткого вывода как для `DesiredSpeed`, так и `Steering` не были изменены — 81 правило в каждом модуле.

Описанный нечёткий контроллер, а также контроллеры GRNDriver, Autopia и обученный нечёткий контроллер были испытаны в заездах на 10 кругов на стандартных трассах TORCS. По итогам заезда время лучшего круга, а также значения максимальной и минимальной скорости каждого контроллера были включены в соответствующие Приложения (см. Приложения А.3-А.6). Для сравнительного анализа были составлены гистограммы времени лучшего круга для избранных трасс (см. Приложения Б.1, Б.2).

По результатам заездов видно, что обученный контроллер проходит трассы значительно лучше, чем базовый нечёткий контроллер. Необученный контроллер, например, не смог завершить прохождение трасс Wheel 2, Alpine 2 и E Track 2, так как данные трассы содержат «шпильки». Видно, что время прохождение круга у контроллера Autopia на трассах Wheel 2 и E Track 2 значительно превосходит среднее время остальных контроллеров по этой же причине.

Стоит обратить внимание на то, что максимальное значение скорости базового нечёткого контроллера не превышает 197 километров в час. Подобную недоработку можно было выявить на стадии разработки контроллера, однако такая проблема не была выявлена у обученного контроллера.

Таким образом, при обучении нечёткой системы сохранились структура и изначальная интерпретируемость системы, а её составные компоненты были значительно оптимизированы, что демонстрируют результаты заездов.

# Заключение

В ходе проделанной работы были выполнены следующие задачи:

1. Проанализированы основные архитектуры нейро-нечётких систем, выделены преимущества и недостатки каждой архитектуры.

2. Разработана и выложена в открытый доступ библиотека Čapek для упрощённой разработки контроллеров гоночного автомобиля в симуляторе TORCS SCR.

3. Разработан контроллер гоночного автомобиля, включающий компоненты на основе нечётких систем, в среде симулятора TORCS.

4. Разработанный контроллер был обучен с применением нечёткой нейронной сети ANFIS на основе данных, полученных с контроллеров Autopia и GRNDriver.

Результаты заездов с использованием разработанного контроллера позволяют сделать вывод, что применённый в данной работе подход является перспективным и потенциально может быть использован для обучения иных систем нечёткого вывода.

Тем не менее в связи с необходимостью дополнительной коррекции обученного контроллера применение пакетной техники обучения нечётких нейронных сетей кажется не лучшим выбором. Вероятно, в дальнейшем стоит применить метод обучения с подкреплением, например, как описано в работе A. Gevaert [34].

Кроме того, кажется интересной идея разработки и применения систем на основе вывода типа Мамдани, например, NEFCON, несмотря на менее точные результаты обученных систем.

# Список литературы


1. *Zadeh L. A.* Fuzzy Logic, Neural Networks, and Soft Computing // Commun. ACM. — New York, NY, USA, 1994. — март. — т. 37, № 3. — с. 77—84. — ISSN 0001-0782. — DOI: 10.1145/175247.175255.
2. *Panda M.* Soft Computing: Concepts and Techniques. — 01.2014. — ISBN 978-93-81159-66-8.
3. *Committee O.-R. A. D.* Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. — 04.2021. — DOI: https://doi.org/10.4271/J3016_202104.
4. *Liu P.*, *Li H.* Fuzzy Neural Network Theory and Application. — WORLD SCIENTIFIC, 2004. — DOI: 10.1142/5493. — eprint: https://www.worldscientific.com/doi/pdf/10.1142/5493.
5. Evolving Rule-Based Explainable Artificial Intelligence for Unmanned Aerial Vehicles / B. M. Keneni [и др.] // IEEE Access. — 2019. — т. 7. — с. 17001—17016. — DOI: 10.1109/ACCESS.2019.2893141.
6. Driving in TORCS Using Modular Fuzzy Controllers / S. Mohammed [и др.] //. — 03.2017. — с. 361—376. — ISBN 978-3-319-55848-6. — DOI: 10.1007/978-3-319-55849-3_24.
7. A modular parametric architecture for the TORCS racing engine / E. Onieva [и др.] //. — 10.2009. — с. 256—262. — DOI: 10.1109/CIG.2009.5286466.
8. Driving Cars by Means of Genetic Algorithms / Y. Sáez [и др.] //. — 09.2008. — с. 1101—1110. — DOI: 10.1007/978-3-540-87700-4_109.
9. Evolving a fuzzy controller for a Car Racing Competition / D. Perez [и др.] // 2009 IEEE Symposium on Computational Intelligence and Games. — 2009. — с. 263—270. — DOI: 10.1109/CIG.2009.5286467.
10. Evolving a TORCS Modular Fuzzy Driver Using Genetic Algorithms / M. Salem [и др.] // Applications of Evolutionary Computation / под ред. K. Sim, P. Kaufmann. — Cham : Springer International Publishing, 2018. — с. 342—357. — ISBN 978-3-319-77538-8.

11. *Wang S.*, *Jia D.*, *Weng X.* Deep Reinforcement Learning for Autonomous Driving. — 2019. — arXiv: 1811.11329 [cs.CV].

12. Deep Reinforcement Learning for Simulated Autonomous Driving / A. Ganesh [и др.] //. — 2016.

13. Using Fuzzy Logic in Automated Vehicle Control / J. Naranjo [и др.] // Intelligent Systems, IEEE. — 2007. — февр. — т. 22. — с. 36—45. — DOI: 10.1109/MIS.2007.18.

14. Throttle and brake pedals automation for populated areas / E. Onieva [и др.] // Robotica. — 2010. — июль. — т. 28. — с. 509—516. — DOI: 10.1017/S0263574709005815.

15. *Singh H.*, *Lone Y. A.* Deep Neuro-Fuzzy Systems with Python. — 1-е изд. — Berkeley, CA : Apress. — с. 260. — ISBN 978-1-4842-5361-8. — DOI: 10.1007/978-1-4842-5361-8.

16. *Nauck D.*, *Kruse R.* Neuro-fuzzy systems for function approximation // Fuzzy Sets and Systems. — 1999. — т. 101, № 2. — с. 261—271. — ISSN 0165-0114. — DOI: 10.1016/S0165-0114(98)00169-9. — Analytical and Structural Considerations in Fuzzy Modeling.

17. *Jang J.-S.* ANFIS: Adaptive-Network-Based Fuzzy Inference System // IEEE Transactions on Systems, Man, and Cybernetics. — 1993. — т. 23, № 3. — с. 665—685. — DOI: 10.1109/21.256541.

18. *Berenji H.*, *Khedkar P.* Learning and tuning fuzzy logic controllers through reinforcements // IEEE Transactions on Neural Networks. — 1992. — т. 3, № 5. — с. 724—740. — DOI: 10.1109/72.159061.

19. *Barto A. G.*, *Sutton R. S.*, *Anderson C. W.* Neuronlike adaptive elements that can solve difficult learning control problems // IEEE Transactions on Systems, Man, and Cybernetics. — 1983. — т. SMC—13, № 5. — с. 834—846. — DOI: 10.1109/TSMC.1983.6313077.

20. *Zadeh L.* Fuzzy sets // Information and Control. — 1965. — т. 8, № 3. — с. 338—353. — ISSN 0019-9958. — DOI: 10.1016/S0019-9958(65)90241-X.

21. *Zadeh L. A.* Fuzzy Logic and Approximate Reasoning (In Memory of Grigore Moisil) // Synthese. — 1975. — т. 30, № 3/4. — с. 407—428. — ISSN 00397857, 15730964.

22. *Brilliant.org.* Artificial Neural Network. — URL: https://brilliant.org/wiki/artificial-neural-network/ (дата обр. 31.05.2021).

23. *Nauck D.* Neuro-Fuzzy Systems: Review And Prospects. — 2000. — июнь.

24. *Abraham A.* Neuro Fuzzy Systems: State-of-the-Art Modeling Techniques // Connectionist Models of Neurons, Learning Processes, and Artificial Intelligen ce / под ред. J. Mira, A. Prieto. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2001. — с. 269—276. — ISBN 978-3-540-45720-6.

25. *Viharos Z.*, *Kis K.* Survey on Neuro-Fuzzy systems and their applications in technical diagnostics and measurement // Measurement. — 2015. — т. 67. — с. 126—136. — ISSN 0263-2241. — DOI: 10.1016/j.measurement.2015.02.001.

26. TORCS: The open racing car simulator / B. Wymann [и др.]. — 2015. — URL: http://torcs.sourceforge.net/ (дата обр. 31.05.2021).

27. *Loiacono D.*, *Cardamone L.*, *Lanzi P. L.* Simulated Car Racing Championship: Competition Software Manual. — 2013. — arXiv: 1304.1672 [cs.AI].

28. *Куклин Д.* Библиотека Čapek для TORCS SCR. — 2021. — (дата обр. 31.05.2021). https://github.com/dikuchan/capek.

29. *Rada-Vilela J.* The FuzzyLite Libraries for Fuzzy Logic Control. — 2018. — URL: https://fuzzylite.com/ (дата обр. 31.05.2021).

30. *Games C. I. in.* Software for the competitions on Computational Intelligence in Games. — (дата обр. 31.05.2021). https://sourceforge.net/projects/cig/.

31. *McKinney W.* Data Structures for Statistical Computing in Python // Proceedings of the 9th Python in Science Conference / под ред. S. van der Walt, J. Millman. — 2010. — с. 56—61. — DOI: 10.25080/Majora-92bf1922-00a. — (дата обр. 31.05.2021).

32. *Power J.* Implementation of ANFIS using the pyTorch framework. — 2019. — URL: https://github.com/GGuedesAB/anfis-pytorch (дата обр. 31.05.2021).

33. PyTorch: An Imperative Style, High-Performance Deep Learning Library / A. Paszke [и др.] // Advances in Neural Information Processing Systems 32 / под ред. H. Wallach [и др.]. — Curran Associates, Inc., 2019. — с. 8024—8035. — URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (дата обр. 31.05.2021).

34. *Gevaert A.*, *Peck J.*, *Saeys Y.* Distillation of Deep Reinforcement Learning Models Using Fuzzy Inference Systems // BNAIC/BENELEARN. — 2019.

| Сенсор | Область значений | Единица | N | Описание |
|---|---|---|---|---|
| `angle` | $[-\pi, \pi]$ | радианы | 1 | Угол между автомобилем и осью трассы |
| `curLapTime` | $[0, +\infty)$ | секунды | 1 | Время текущего круга |
| `damage` | $[0, +\infty)$ | | 1 | Урон автомобиля |
| `distFromStart` | $[0, +\infty)$ | метры | 1 | Расстояние, пройденное по трассе |
| `distRaced` | $[0, +\infty)$ | метры | 1 | Общее пройденное расстояние |
| `focus` | $[0, 200]$ | метры | 5 | Расстояния до края трассы<br>5 направлений с разницей в 1° |
| `fuel` | $[0, +\infty)$ | | 1 | Текущий уровень топлива |
| `gear` | $\{-1, 0, \ldots, 6\}$ | | 1 | Текущая передача |
| `lastLapTime` | $[0, +\infty)$ | секунды | 1 | Время предыдущего круга |
| `opponents` | $[0, 200]$ | метры | 36 | Расстояния до ближайших оппонентов |
| `racePos` | $\{1, 2, \ldots, N\}$ | | 1 | Место участника гонки |
| `rpm` | $[0, +\infty)$ | обороты в минуту | 1 | Число оборотов в минуту |
| `speedX` | $(-\infty, +\infty)$ | километры в час | 1 | Скорость вдоль продольной оси |
| `speedY` | $(-\infty, +\infty)$ | километры в час | 1 | Скорость вдоль поперечной оси |
| `speedZ` | $(-\infty, +\infty)$ | километры в час | 1 | Скорость вдоль оси $Z$ |
| `track` | $[0, 200]$ | метры | 19 | Расстояния до края трассы |
| `trackPos` | $(-\infty, +\infty)$ | | 1 | Положение по отношению к оси трассы |
| `wheelSpinVel` | $[0, +\infty]$ | радианы в секунду | 4 | Скорость вращения колёс |
| `z` | $(-\infty, +\infty)$ | метры | 1 | Положение по отношению к оси $Z$ |

**Приложение А.1:** Описание доступных сенсоров TORCS SCR

| Контроллер | Область значений | Описание |
|---|---|---|
| `accel` | $[0, 1]$ | Сила нажатия на педаль газа |
| `brake` | $[0, 1]$ | Сила нажатия на педаль тормоза |
| `clutch` | $[0, 1]$ | Сила нажатия на педаль сцепления |
| `gear` | $\{-1, 0, \ldots, 6\}$ | Значение передачи |
| `steering` | $[-1, 1]$ | Направление поворота рулевого колеса |
| `focus` | $[-90, 90]$ | Направление взгляда |
| `meta` | $\{0, 1\}$ | Запрос на перезапуск сервера |

**Приложение А.2:** Описание доступных команд управления TORCS SCR

| Трасса | Время круга, м | Макс. скорость, $\frac{\text{км}}{\text{ч}}$ | Мин. скорость, $\frac{\text{км}}{\text{ч}}$ |
|---|---|---|---|
| Ruudskogen | 1:17:61 | 209 | 77 |
| Wheel 2 | 2:15:41 | 247 | 68 |
| Alpine 1 | 2:43:75 | 213 | 54 |
| Alpine 2 | 1:46:32 | 193 | 56 |
| Brondehach | 1:36:09 | 206 | 48 |
| Corkscrew | 1:43:46 | 190 | 26 |
| E Track 2 | 2:47:71 | 202 | 20 |
| E Track 3 | 1:49:64 | 224 | 47 |
| E Road | 1:17:30 | 235 | 63 |
| Forza | 1:56:79 | 261 | 64 |
| CG Track 1 | 0:47:13 | 211 | 92 |
| CG Track 2 | 1:06:88 | 232 | 111 |
| CG Track 3 | 1:14:91 | 191 | 96 |

**Приложение А.3:** Результаты обученного контроллера

| Трасса | Время круга, м | Макс. скорость, $\frac{км}{ч}$ | Мин. скорость, $\frac{км}{ч}$ |
|---|---|---|---|
| Ruudskogen | 1:14:87 | 225 | 76 |
| Wheel 2 | 7:20:80 | 214 | 19 |
| Alpine 1 | 8:10:50 | 219 | 19 |
| Alpine 2 | 1:44:48 | 225 | 69 |
| Brondehach | 1:24:80 | 237 | 67 |
| Corkscrew | 1:25:47 | 218 | 54 |
| E Track 2 | 5:13:73 | 227 | 19 |
| E Track 3 | 1:35:36 | 241 | 81 |
| E Road | 1:06:37 | 255 | 100 |
| Forza | 5:50:79 | 262 | 19 |
| CG Track 1 | 0:41:76 | 240 | 107 |
| CG Track 2 | 0:58:30 | 251 | 140 |
| CG Track 3 | 1:16:14 | 211 | 93 |

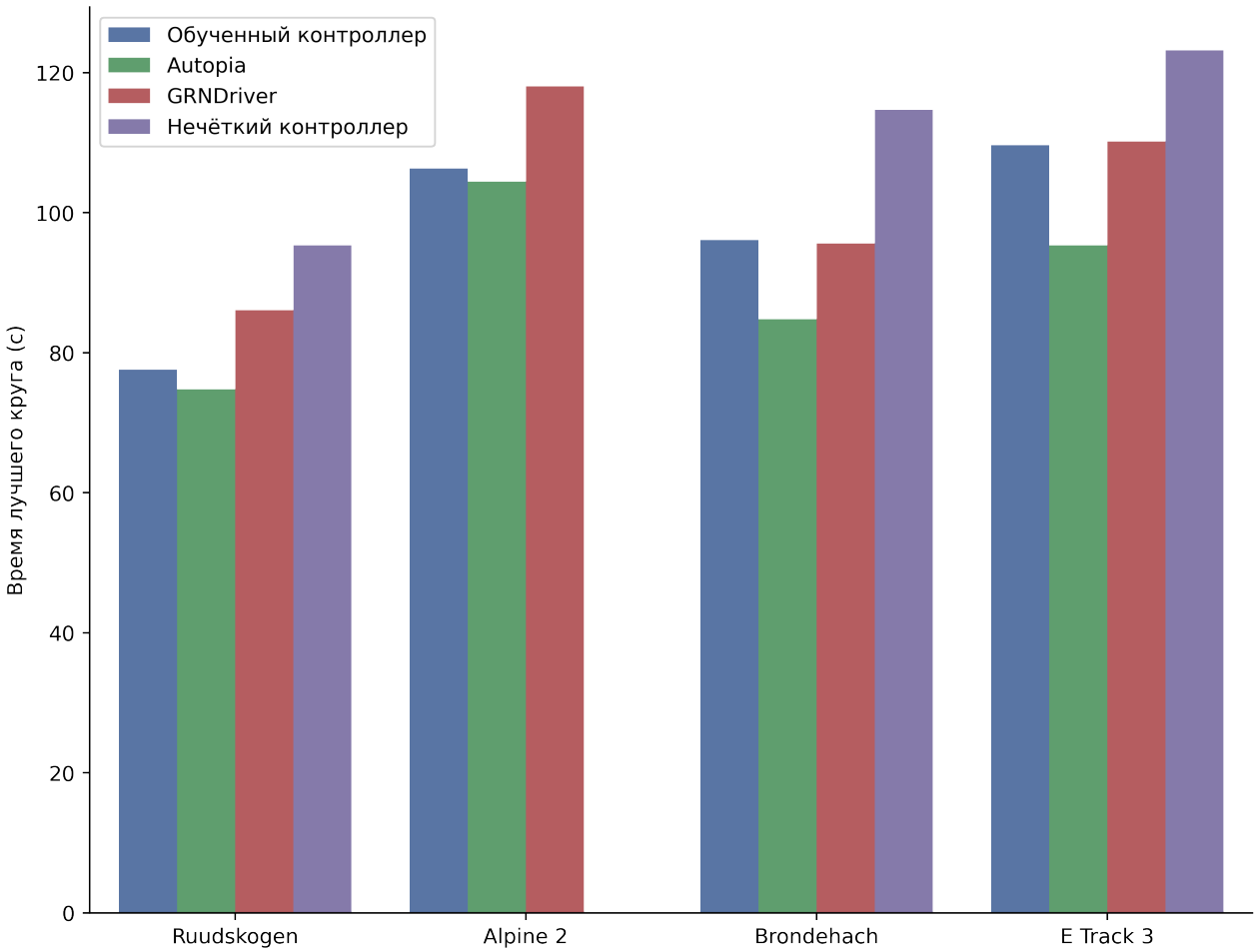
**Приложение А.4:** Результаты контроллера Autopia

| Трасса | Время круга, м | Макс. скорость, $\frac{км}{ч}$ | Мин. скорость, $\frac{км}{ч}$ |
|---|---|---|---|
| Ruudskogen | 1:26:08 | 191 | 78 |
| Wheel 2 | 2:28:67 | 215 | 39 |
| Alpine 1 | 2:52:59 | 212 | 29 |
| Alpine 2 | 1:58:08 | 189 | 54 |
| Brondehach | 1:35:62 | 224 | 59 |
| Corkscrew | 1:38:88 | 187 | 36 |
| E Track 2 | 2:26:80 | 212 | 19 |
| E Track 3 | 1:50:20 | 212 | 40 |
| E Road | 1:17:55 | 238 | 39 |
| Forza | 2:08:62 | 201 | 59 |
| CG Track 1 | 0:45:07 | 222 | 112 |
| CG Track 2 | 1:04:60 | 239 | 122 |
| CG Track 3 | 1:23:42 | 189 | 70 |

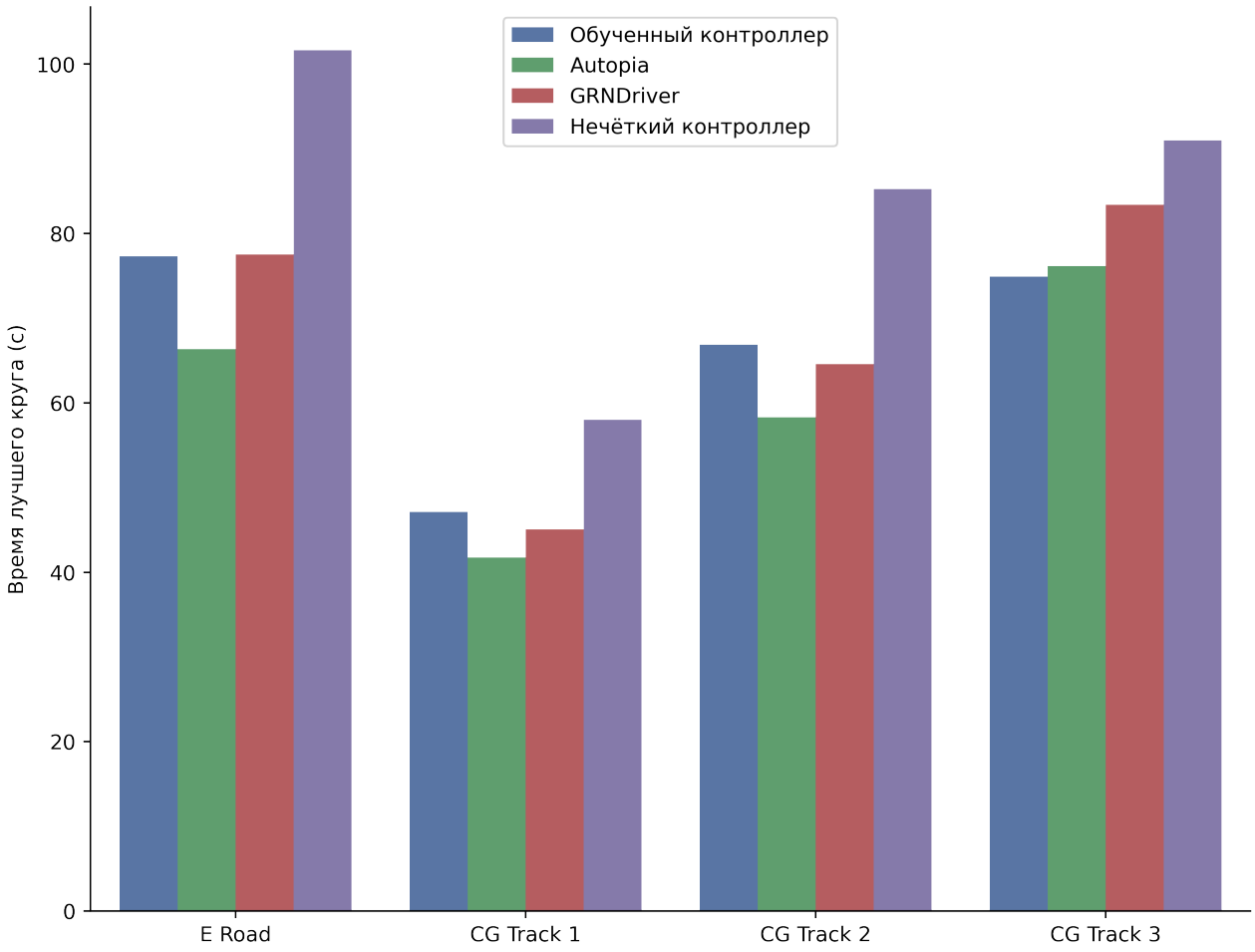
**Приложение А.5:** Результаты контроллера GRNDriver

| Трасса | Время круга, м | Макс. скорость, $\frac{км}{ч}$ | Мин. скорость, $\frac{км}{ч}$ |
|---|---|---|---|
| Ruudskogen | 1:35:34 | 194 | 83 |
| Wheel 2 | - | - | - |
| Alpine 1 | 3:18:69 | 197 | 65 |
| Alpine 2 | - | - | - |
| Brondehach | 1:54:70 | 196 | 75 |
| Corkscrew | 5:08:76 | 171 | 0 |
| E Track 2 | - | - | - |
| E Track 3 | 2:03:20 | 196 | 74 |
| E Road | 1:41:65 | 197 | 61 |
| Forza | 2:22:13 | 197 | 82 |
| CG Track 1 | 0:58:01 | 189 | 89 |
| CG Track 2 | 1:25:25 | 197 | 75 |
| CG Track 3 | 1:31:02 | 177 | 69 |

**Приложение А.6:** Результаты нечёткого контроллера




**Приложение Б.1:** Сравнение времени лучшего круга для разных контроллеров




**Приложение Б.2:** Сравнение времени лучшего круга для разных контроллеров