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

ПАНЧУК Дмитрий Вадимович

АДАПТИВНОЕ УПРАВЛЕНИЕ ЦИФРОВЫМ ДВОЙНИКОМ МОБИЛЬНОГО РОБОТА

Дипломная работа

Научный руководитель:
кандидат физико-
математических наук, доцент
И.С. Козловская

Допущена к защите
«____»_____20___г.
Заведующий кафедрой компьютерных технологий и систем
доктор педагогических наук, кандидат физико-
математических наук, профессор В.В. Казачёнок

Минск, 2025

ОГЛАВЛЕНИЕ

РЕФЕРАТ

Дипломная работа: 64 с., 44 рис., 3 таблицы, 24 источника.

Ключевые слова: ЦИФРОВОЙ ДВОЙНИК, МОБИЛЬНЫЙ РОБОТ, СИСТЕМЫ УПРАВЛЕНИЯ, MATLAB.

Цель дипломной работы – исследовать возможности технологии цифрового двойника для создания систем управления мобильным роботом.

Методы исследования: анализ и исследование технологии, математическое описание систем, моделирование.

В результате работы был разработан цифровой двойник мобильного робота RoboCake, а также реализованы системы управления этим двойником с использованием средств MATLAB Simulink. Созданная модель достаточно точно воспроизводит поведение реального устройства, что позволяет проводить виртуальные испытания, отладку и оптимизацию алгоритмов управления без необходимости использования физического прототипа.

Область возможного применения: промышленное производство, логистика, военная сфера, пожарные структуры, исследовательская деятельность и другие.

Социально-экономическая значимость: данной работы обусловлена высокой трудоёмкостью и значительными финансовыми затратами, характерными для разработки и настройки систем управления мобильными роботами. Предложенное решение позволяет существенно упростить этот процесс за счёт использования цифрового двойника, который может быть адаптирован под любую робототехническую платформу и интегрирован в существующие или разрабатываемые системы управления, тем самым снижая издержки и ускоряя цикл разработки.

РЭФЕРАТ

Дыпломная работа: 64 с., 44 мал., 3 табліцы, 24 крыніцы.

Ключавыя словы: лічбавы двайнік, мабільны робат, сістэмы кіравання, MATLAB.

Мэта дыпломнай працы – даследаваць магчымасці тэхналогіі лічбавага двайніка для стварэння сістэм кіравання мабільным робатам.

Метады даследавання: Аналіз і даследаванне тэхналогіі, матэматычнае апісанне сістэм, мадэляванне.

У выніку працы быў распрацаваны лічбавы двайнік мабільнага робата RoboCake, а таксама рэалізаваны сістэмы кіравання гэтым двайніком з выкарыстаннем сродкаў MATLAB Simulink. Створаная мадэль досыць дакладна прайгравае паводзіны рэальнага прылады, што дазваляе праводзіць віртуальныя выпрабаванні, адладку і аптымізацыю алгарытмаў кіравання без неабходнасці выкарыстання фізічнага прататыпа.

Вобласць магчымага прымянення: прамысловая вытворчасць, лагістыка, ваенная сфера, пажарныя структуры, даследчая дзейнасць і іншыя.

Сацыяльна-эканамічная значнасць: дадзенай працы абумоўлена высокай працаёмкасцю і значнымі фінансавымі выдаткамі, характэрнымі для распрацоўкі і налады сістэм кіравання мабільнымі робатамі. Прапанаванае рашэнне дазваляе істотна спрасціць гэты працэс за кошт выкарыстання лічбавага двайніка, які можа быць адаптаваны пад любую робататэхнічных платформу і інтэграваны ў існуючыя або распрацоўваюцца сістэмы кіравання, тым самым зніжаючы выдаткі і паскараючы цыкл распрацоўкі.

SUMMARY

Diploma thesis: 64 pp., 44 pics., 3 tables, 24 sources.

Keywords: DIGITAL TWIN, MOBILE ROBOT, CONTROL SYSTEMS, MATLAB.

The purpose of the thesis – investigate the possibilities of digital twin technology to create control systems for mobile robot.

Research methods: analysis and research of technology, mathematical description of systems, modelling.

As a result of the work, a digital twin of the mobile robot RoboCake was developed, and control systems of this twin were implemented using MATLAB Simulink. The created model accurately reproduces the behaviour of a real device, which allows virtual testing, debugging and optimization of control algorithms without the need to use a physical prototype.

The field of possible practical application: industrial production, logistics, military sphere, fire structures, research activities and others.

Socio-economic significance: this work is due to the high labour intensity and significant financial costs typical for the development and configuration of control systems for mobile robots. The proposed solution allows us to significantly simplify this process by using a digital twin that can be adapted to any robotics platform and integrated into existing or developing control systems, thus reducing costs and accelerating the development cycle.

ВВЕДЕНИЕ

В современном мире наблюдается активное развитие технологий, оказывающих значительное влияние на все сферы человеческой жизни. Достижения в таких областях, как робототехника, искусственный интеллект, компьютерное зрение, анализ больших данных, квантовые вычисления и биотехнологии, не только упрощают повседневные задачи, но и открывают новые перспективы для научных исследований, промышленности и медицины.

Темпы прогресса столь высоки, что технологии, которые считались инновационными несколько лет назад, сегодня становятся обыденными. Это требует от общества гибкости и постоянной адаптации к изменениям, а также развития навыков, позволяющих эффективно использовать новые технологии для решения возникающих проблем.

В этих условиях робототехника, предлагает решения для автоматизации сложных процессов в различных сферах. В новом отчете World Robotics 2024 зафиксировано 4 281 585 устройств, работающих на заводах по всему миру, что на 10 % больше, чем в прошлом году. Ежегодное количество установок превысило полмиллиона единиц третий год подряд. По регионам: 70 % всех новых роботов 2023 года были установлены в Азии, 17 % в Европе и 10 % в Северной и Южной Америке [24]. Исходя из статистики, представленной на рисунке 1, можно сделать вывод, что активный рост использования роботов наблюдается каждый год, что подтверждает актуальность исследований в области робототехники.

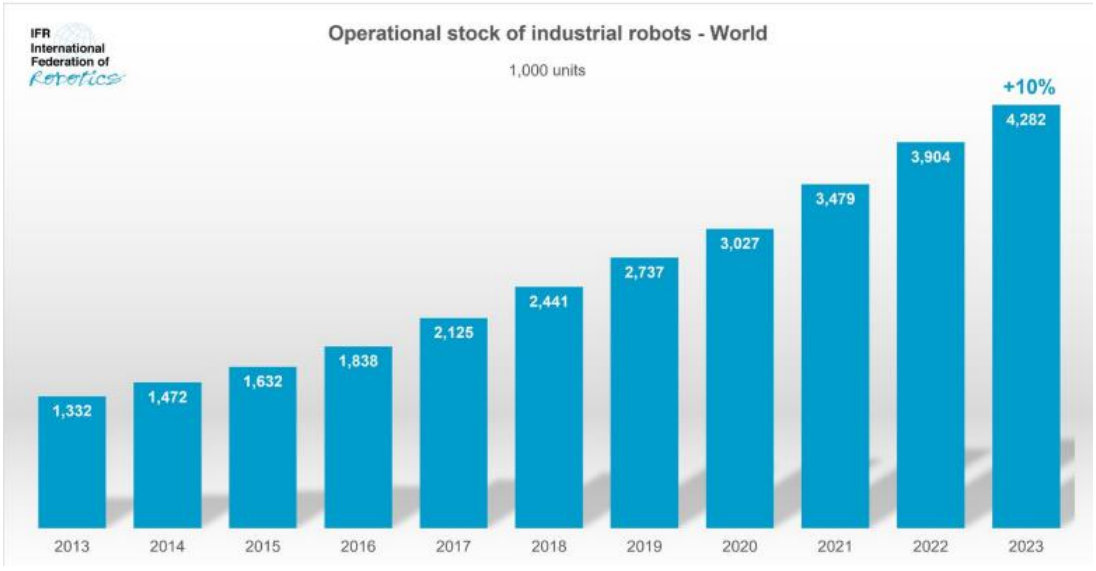

Рисунок 1 – Тренд роста числа используемых роботов в мире [24]

В свою очередь мобильная робототехника представляет собой направление робототехники, занимающееся разработкой и применением автономных или полуавтономных роботов, способных перемещаться в пространстве без привязки к стационарным объектам. Такие роботы оснащаются системами навигации, датчиками и средствами управления, что позволяет им взаимодействовать с окружающей средой и адаптироваться к её изменениям [24].

Однако разработка мобильного робота представляет собой сложный и дорогостоящий процесс, который включает в себя проектирование механической конструкции, создание систем управления, интеграцию сенсоров и программного обеспечения, а также тестирование в различных условиях. Эти этапы требуют значительных временных, финансовых и интеллектуальных ресурсов, что ограничивает доступность использования мобильных роботов.

В связи с этим актуальным является упрощение процесса разработки мобильных роботов с помощью новых технологий, таких как цифровые двойники, и универсальные алгоритмы управления. Эти технологии могут снизить затраты, ускорить процесс разработки и сделать мобильную робототехнику более доступной для широкого круга специалистов.

Таким образом цель данной дипломной работы: «Исследовать возможности технологии цифрового двойника для создания систем управления мобильным роботом».

ГЛАВА 1
ПОНЯТИЕ ЦИФРОВОГО ДВОЙНИКА МОБИЛЬНОГО РОБОТА И СИСТЕМ УПРАВЛЕНИЯ МОБИЛЬНЫМ РОБОТОМ

1.1 Технология создания цифрового двойника, её основные понятия и примеры применения

Технология создания цифрового двойника, берёт своё начало в концепции «Индустрии 4.0», описанной Клаусом Швабом в 2016 году [16].

«Индустрия 4.0» (или четвёртая промышленная революция) — это этап развития технологий, характеризующийся автоматизацией производственных и промышленных процессов с использованием таких технологий, как промышленные интернет предметы (IoT), аналитика больших данных, искусственный интеллект, робототехника и автономные системы [20]. Компоненты «Индустрии 4.0» представлены на рисунке 1.1.

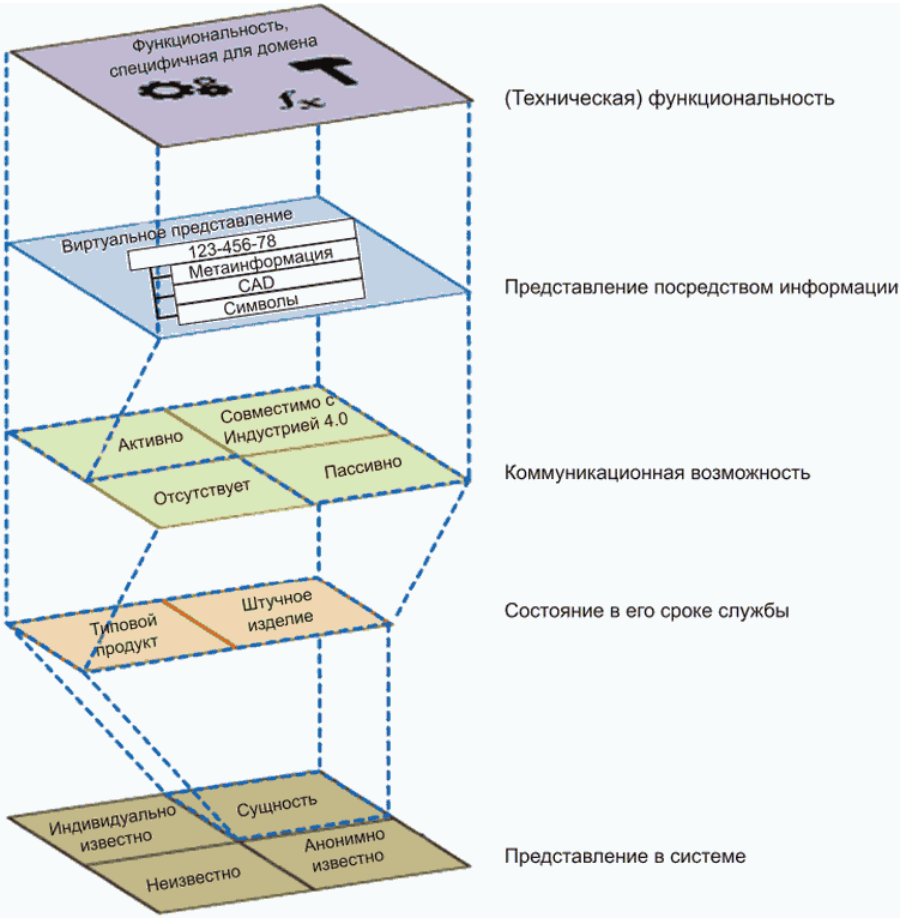

Рисунок 1.1 – Компоненты Индустрии 4.0 [12]

Целью «Индустрии 4.0» является рост производственных мощностей и обеспечение гибкого, ориентированного на потребности клиента, производства, а также снижение затрат на эксплуатацию и техническое обслуживание производственных ресурсов.

Для изучения и применения технологии создания цифрового двойника, необходимо дать определения её основным понятиям.

Цифровой двойник - система, состоящая из цифровой модели субъекта реального мира и информационных связей с этим субъектом и (или) его составными частями [6].

Цифровая модель - система математических и компьютерных моделей, а также электронных документов изделия, описывающая структуру, функциональность и поведение разрабатываемого или эксплуатируемого изделия на различных стадиях жизненного цикла [6].

Цифровой двойник содержит модели своих «данных» (геометрия, структура), своей функциональности (обработка данных, поведение) и своих коммуникационных интерфейсов. Он представляет собой виртуальный аналог реального объекта, который используется для проверки и подтверждения данных, а также для обеспечения работы приложений в физическом мире [18]. Благодаря сохранению согласованности между виртуальным и реальным мирами, мы можем переносить разработанные системы из киберпространства в физический мир. Примеры использования цифровых двойников в различных сферах представлены на рисунке 1.2.

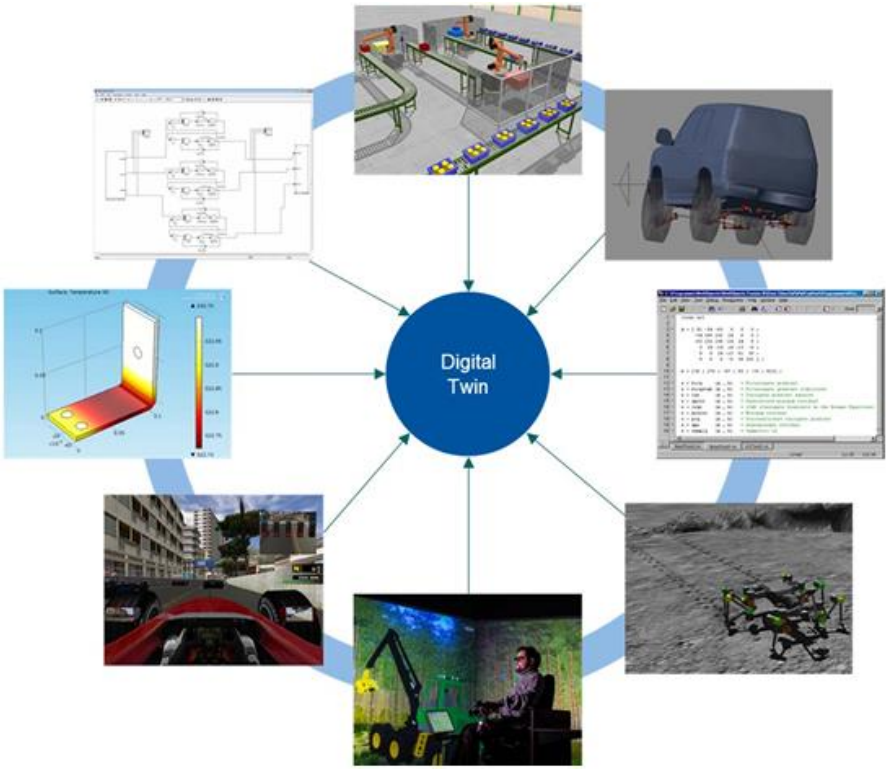

Рисунок 1.2 – Примеры использования цифровых двойников [18]

Наряду с понятием цифрового двойника, вводят также понятие следящей тени. Следящая тень представляет собой компонент цифрового двойника, который в режиме реального времени отслеживает и воспроизводит текущее состояние, на основе данных поступающих от него [6].

Следящая тень является связующим звеном между физическим объектом и его цифровым представлением, гарантируя актуальность данных для анализа и управления.

Эффективным инструментом для проверки и взаимодействия с цифровым двойником служат цифровые испытательные полигоны.

Цифровой испытательный полигон — система с техническими средствами, программным обеспечением, методиками, организацией и персоналом для испытаний цифровой модели объекта [6].

Цифровые испытательные полигоны эффективно создают и тестируют цифровые двойники. Цифровой двойник системы включает моделируемую систему, её датчики и исполнительные механизмы, а также алгоритмы обработки данных. Всё это функционирует в цифровом двойнике окружающей среды.

Для создания и использования цифровых двойников требуется симулятор с необходимыми характеристиками. Симулятор — это программа для построения и исполнения моделей, воспроизводящих динамическое поведение системы [6].

1.2 Анализ и сравнение популярных симуляторов

Наиболее популярными симуляторами в настоящее время являются MATLAB Simulink, ANSYS Twin Builder и OpenModelica.

MATLAB Simulink — это среда для создания блок-схем, используемая для проектирования систем с многодоменными моделями, моделирования перед переходом на аппаратное обеспечение и развёртывания без написания кода. Она интегрируется с MATLAB и позволяет разрабатывать, тестировать и оптимизировать цифровые двойники [23].

Ввиду своей популярности и большого сообщества пользователей, данный симулятор имеет значительные преимущества. Во-первых, главным преимуществом является поддержка механических, электрических, гидравлических, тепловых и других типов систем, что подтверждается наличием богатого набора встроенных компонентов и моделей. Во-вторых, MATLAB Simulink имеет понятный блок-схемный вид программ, с возможностью интеграции пользовательских алгоритмов (даже реализованных в других средах разработки). Так же MATLAB Simulink имеет

возможность генерации кода для встраиваемых систем и работы с аппаратным обеспечением.

К недостаткам этого симулятора можно отнести: трудность в изучении, требовательность к вычислительным ресурсам и стоимость подписки (имеется бесплатный пробный период).

Данный симулятор широко используется в автомобильной промышленности, аэрокосмической отрасли, энергетике и, что важно в рамках данной работы, в разработке цифровых двойников в машиностроении и робототехнике.

ANSYS Twin Builder — открытое решение, позволяющее инженерам, создавать цифровые двойники на основе моделирования с помощью Hybrid Analytics [14].

Главным преимуществом данного симулятора, является его специализация на создании и использовании цифровых двойников. Возможность соединения с системами IoT для получения данных в реальном времени и возможность поддержки высокоточных физических моделей, являются не менее весомыми преимуществами.

Важным недостатком является сложность в интеграции с другими инструментами моделирования. Наряду с этим недостатком, можно выделить также: высокую стоимость лицензии, высокий порог входа и ограниченность в работе с исходным кодом.

Основными сферами применения являются: создание промышленного оборудования, автомобилей и авиационной техники.

OpenModelica — это среда моделирования и симуляции на базе Modelica1 с открытым исходным кодом, предназначенная для промышленного и академического использования [22].

К преимуществам можно отнести: открытый код, многодоменность и активная поддержка сообществом. Однако данный симулятор обладает наибольшим из троих количеством недостатков: симулятор не всегда справляется с большими и сложными моделями, пользовательский интерфейс менее удобен и интуитивен по сравнению с коммерческими инструментами. Модели с высокой степенью нелинейности, как например модель цифрового двойника мобильного робота, могут быть трудными для симуляции, что приводит к увеличению времени расчетов.

Исходя из данного описания была составлена таблица 1, описывающая плюсы и минусы каждого симулятора.

Таблица 1 – сравнение популярных симуляторов

Критерий	MATLAB Simulink	ANSYS Twin Builder	OpenModelica
Поддержка многодоменных моделей	+	-	+
Удобство интерфейса	+	-	-
Интеграция с другими инструментами	+	-	+
Генерация кода для встраиваемых систем	+	-	-
Взаимодействие с физическим оборудованием (IoT, встраиваемые системы)	+	+	-
Точность при работе с физическими моделями	+	+	-
Гибкость в работе с исходным кодом	-	-	+
Скорость расчётов для больших моделей	+	+	-
Поддержка сообществом (большое количество готовых открытых решений)	+	-	+
Подробная документация и обучающие материалы	+	+	-
Применение в промышленности	+	+	-
Платная лицензия	+	+	-
Наличие бесплатного периода для студентов	+	+	-

1.3 Понятие систем управления мобильным роботом, их классификация и примеры

Системы управления мобильными роботами (СУМР) — это совокупность аппаратных и программных компонентов, обеспечивающих выполнение задач роботом в различных условиях.

Рассмотрим классификацию СУМР по наиболее важным признакам: по архитектуре управления, по методам управления, по способу взаимодействия с окружающей средой и по уровню автономности [9].

По архитектуре управления СУМР можно разделить на: централизованные системы, децентрализованные системы и иерархические системы.

В централизованных системах, принятие решений осуществляется в одном узле. Данный подход является простым в реализации, но обладает слабой устойчивостью к сбоям.

В децентрализованных системах, робот разделяется на независимые модули, которые самостоятельно принимают решения, и при необходимость обмениваются информацией с другими модулями. Данный подход обладает сильной устойчивостью, но имеет трудности в реализации координации между роботами.

Иерархические системы управления разделяют на уровни (высокие, средние и низкие), где каждый уровень отвечает за свои задачи. Например: навигация, обработка показаний сенсоров, управление движением. Нижние уровни координируются, более высокими, что упрощает процесс координации, но понижает устойчивость, в связи с рисками выхода верхних уровней из строя.

По методам управления СУМР можно разделить на: детерминированные, управляемые на основе моделей, реактивно управляемые и управляемые поведенческой моделью.

В детерминированных методах управления, поведение робота заранее определяется чётким алгоритмом. Данный подход отлично проходит для простых и статичных задач, которые не требуют адаптации робота в случае изменения обстановки. Однако данный подход требует чёткого планирования и затрат на разработку алгоритма, что не подходит для более сложных задач, в которых важна возможность робота подстраиваться под новые условия.

В методах, основанных на модели, программные компоненты СУМР содержат математические модели окружающей среды и/или собственные кинематические и динамические модели робота, что позволяет просчитывать поведение робота наперёд.

Реактивно управляемые системы позволяют принимать решения по управлению в реальном времени, без сложного предварительного анализа среды. Данный подход проходит для изменяющихся задач (например движение по линии), так как позволяет обеспечить адаптивность робота, но в случае необходимости чёткого выполнения задач (без права на ошибку), могут возникнуть ситуации, когда робот поведёт себя непредсказуемо.

При применении поведенческой модели, в методах в СУМР закладывается определённый «поведенческий модуль», который подходит для решения конкретной задачи. Например: избегание препятствий или следование за целью.

По алгоритмам навигации и локализации СУМР можно разделить на: использующие сенсоры робота, использующие взаимодействие с облачной инфраструктурой и мультироботные системы.

Управление в системах, использующих сенсоры робота осуществляется исходя из информации, полученной с сенсоров робота. Примеры сенсоров: лидары, камеры, ультразвуковые датчики, датчики освещённости, GPS и др.

В системах, взаимодействующих с облачной инфраструктурой, управление робота основано на информации, полученной от стороннего наблюдателя (например камера видеонаблюдения).

Мультироботные системы используют несколько роботов, обменивающихся информацией, что позволяет получать больше информации об обстановке, решать более сложные задачи и лучше управлять роботами.

По уровню автономности СУМР разделяют на: ручные, автономные и полуавтономные.

В ручных системах управление осуществляется оператором через интерфейс или контроллер.

В автономных системах управление осуществляется заложенным в программную часть алгоритмом, что позволяет роботу действовать без помощи оператора.

В полуавтономных системах управление осуществляется, как и в автономных системах, однако в случае необходимости оператор может осуществлять ручное управление.

Примерами классических систем управления мобильным роботом можно считать:

1. Системы ручного управления. По приведённой выше классификации, такие системы зачастую являются ручными реактивными централизованными системами. Основой таких систем является оператор (операторы), который решает, как изменить положение робота, пользуясь определённым пультом управления.
2. Системы, основанные на ПИД регуляторах. По приведённой выше классификации, такие системы зачастую являются автономными централизованными реактивными системами, использующими сенсоры робота для принятия решений. Основой таких систем является ПИД регулятор (регуляторы), который на основании отклонения системы от “идеального” состояния вырабатывает управляющее воздействие, влияющее на поведение робота.

3. Системы, основанные на построенных алгоритмах. Классификация таких систем проводится на основе конкретного алгоритма. Конкретным примером таких систем можно рассмотреть системы, основанные на обученной нейронной сети, входными данными которой являются показания лидара, а выходными управляющее воздействие, применяемое к роботу.

1.4 Понятие цифрового двойника мобильного робота и требования предъявляемые к нему

На основании приведённой выше информации, необходимо получить определение цифрового двойника мобильного робота и сформировать ряд требований к его реализации.

Цифровой двойник мобильного робота — это компьютерная модель, построенная на основе математического описания реального робота, которая с высокой точностью воспроизводит его физические, кинематические и динамические характеристики. Такой двойник позволяет проводить исследования и тестирование информационных связей мобильного робота с окружающей средой без необходимости использования физического устройства.

Требования к цифровому двойнику мобильного робота

1. Точность моделирования физических характеристик. Цифровой двойник должен корректно воспроизводить геометрию, массу, инерционные параметры, а также поведение приводов, сенсоров и других компонентов реального робота.

2. Реалистичное взаимодействие с виртуальной средой. Цифровой двойник должен обеспечивать достоверное моделирование контакта с поверхностями, учитывать трение, препятствия, динамические изменения окружающей среды.

3. Гибкость в выборе систем управления. Должна быть возможность интеграции и тестирования различных систем управления, а также их переноса на физический робот с минимальными изменениями.

4. Поддержка работы в реальном времени. Цифровой двойник должен обеспечивать возможность имитации работы в реальном времени для проведения тестирования систем управления.

5. Интерактивность и визуализация. Необходимо наличие удобного интерфейса для мониторинга состояний цифрового двойника, визуализации его работы, а также возможности интерактивного взаимодействия с пользователем.

6. Возможность сбора и анализа данных. Цифровой двойник должен включать инструменты для записи, хранения и анализа данных, полученных в ходе моделирования, что позволит совершенствовать системы управления и проводить их диагностику.

7. Масштабируемость и модульность. Архитектура цифрового двойника должна быть гибкой, поддерживая расширение функционала, добавление новых компонентов и интеграцию с различными робототехническими системами.

Приведённый перечень требований позволит создать цифровой двойник мобильного робота, максимально приближенный к его физическому аналогу и пригодный для широкого спектра исследований и тестирования.

1.5 Постановка цели и задач

Проводя анализ различных источников, были сделаны следующие выводы:

1. Технология создания цифрового двойника является актуальным инструментом, который можно применять во многих областях, в частности в разработке систем управления мобильными роботами.
2. Анализ популярных симуляторов, позволил сделать выбор наиболее подходящего варианта для реализации цифрового двойника мобильного робота – MATLAB Simulink.
3. Удачным выбором для проверки возможностей технологии цифрового двойника для создания системы управления мобильным роботом являются классические системы управления (ручное управление, управление при помощи ПИД-регуляторов и управление на основе заранее обученной нейронной сети), так как они являются простыми в реализации и позволяют собрать большой объём исследовательской информации.

Исходя из данных выводов была поставлена следующая цель: «Исследовать возможности технологии цифрового двойника для создания систем управления мобильным роботом».

Для достижения поставленной цели необходимо решить следующие задачи:

1. Описать физическую модель мобильного робота и математическую модель его движения.
2. Описать системы управления мобильным роботом.
3. Создать цифрового двойника по описанной модели.

4. Реализовать системы управления мобильным роботом, провести их настройку и запуск в визуальном стенде.
5. Сделать выводы о проделанной работе.

ГЛАВА 2
МАТЕМАТИЧЕСКОЕ ОПИСАНИЕ МОБИЛЬНОГО РОБОТА И ЕГО СИСТЕМ УПРАВЛЕНИЯ

2.1 Физическая модель мобильного робота RoboCake

В качестве физической модели для исследования был разработан мобильный робот RoboCake. Изображение разработанного робота представлено на рисунке 2.1.

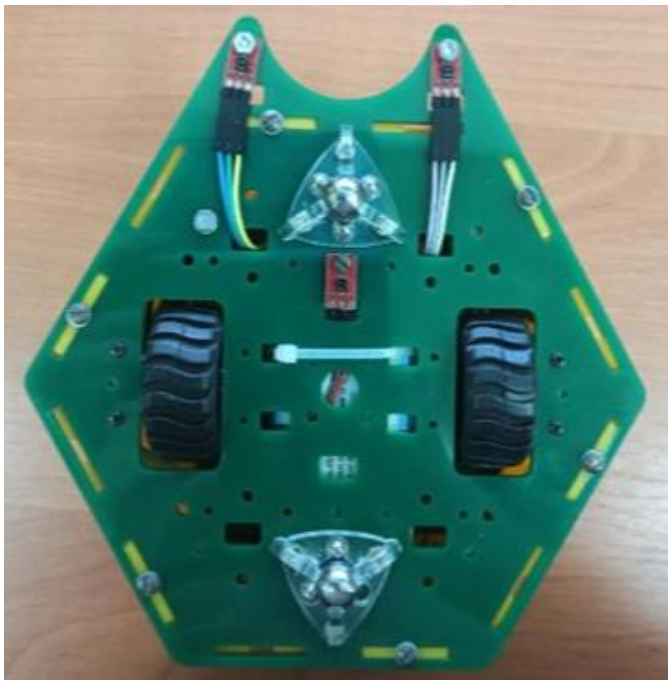

Рисунок 2.1 – Изображение мобильного робота RoboCake [8]

Данный мобильный робот снабжён двумя колёсами, приводимыми в движение дифференциальным приводом. Также он оборудован инкрементными энкодерами и тремя датчиками света. Для повышения устойчивости робот имеет две шарнирные опоры [8].

Физические параметры мобильного робота RoboCake:

- длина корпуса 17,1 см;
- ширина корпуса 16,9 см;
- радиус колёс 2,5 см;
- ширина колеи 8,5 см;
- масса корпуса 400 г.

Схема расположения датчиков света представлена на рисунке 2.2.

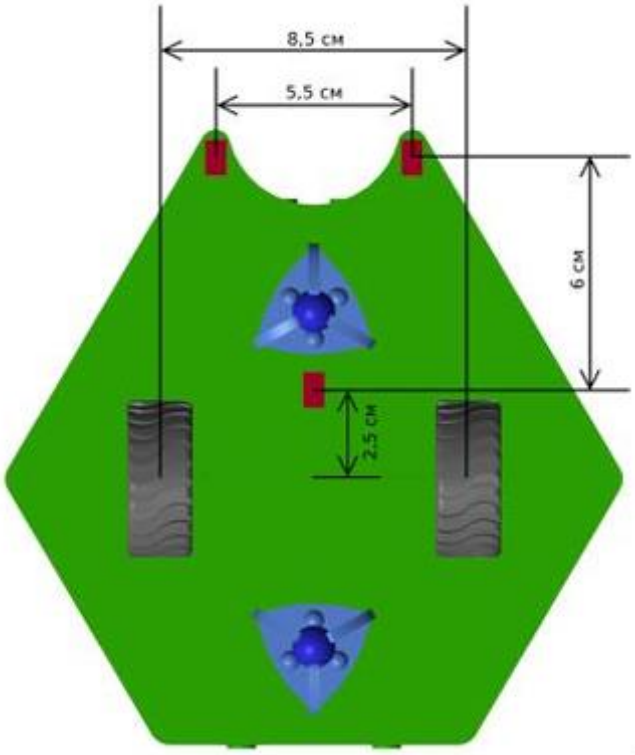

Рисунок 2.2 – Схема расположения датчиков света [8]

В качестве датчиков света были выбраны датчики серии QRE1113, имеющие следующие основные для нас характеристики [5]:

- количество каналов: 1;
- количество контактов: 4;
- типовое время нарастания: 20 мкс;
- длина: 3,6 мм;
- ширина: 2,9 мм;
- высота: 1.7 мм;
- расстояние срабатывания: 1 мм;
- метод считывания: отражательный;
- длина волны: 940 нм;
- вес: 0.5 г;

Установленный электродвигатель (TT Geared Motor), имеет следующие основные характеристики:

- Диапазон рабочих напряжений: 3 - 7,5 В;
- Скорость холостого хода при напряжении 7,5 В: 160 об/мин;
- Режим вывода: 2 стороны.

2.2 Математическая модель движения мобильного робота

Описание математической модели движения робота производится с целью представления процесса его движения в виде системы уравнений и зависимостей. Они связывают параметры движения робота с физическими характеристиками робота.

Для начала рассчитаем диапазон скоростей колёс $[-\omega_{max}, \omega_{max}]$. ω_{max} исходя из характеристик электродвигателя можно найти следующим образом:

$$\omega_{max} = \frac{7{,}5\text{В} \times 160 \frac{\text{об}}{\text{мин}}}{6\text{В}} = 200 \frac{\text{об}}{\text{мин}} = 20.9439 \frac{\text{рад}}{\text{с}}.$$

Таким образом получаем, что значения угловой скорости каждого колеса могут принадлежать диапазону $[-20.9439 \frac{\text{рад}}{\text{с}}, 20.9439 \frac{\text{рад}}{\text{с}}]$.

Мобильный робот RoboCake, как было сказано выше, имеет дифференциальный привод, который позволяет изменять угловую скорость вращения каждого из колёс независимо.

Введём следующие обозначения:

- $\omega_L(t)$ и $\omega_R(t)$ — угловая скорость левого и правого колёс в момент времени t соответственно;
- $\vartheta(t)$ — линейная скорость робота в момент времени t;
- x = x(t) и y = y(t) – координаты робота в момент времени t (в декартовой системе координат);
- θ = θ(t) — угол ориентации робота в момент времени t.
- r = 2,5 см. — радиус колёс;
- L = 8,5 см. — ширина колеи;

Для начала выразим среднюю линейную скорость робота ϑ и угловую скорость центра робота ω:

$$\vartheta = r \frac{\omega_R + \omega_L}{2}, \tag{2.1}$$

$$\omega = r \frac{\omega_R - \omega_L}{L}. \tag{2.2}$$

Используя формулы (2.1) и (2.2), запишем дифференциальные уравнения для измерения x, y, θ в момент времени t:

$$\begin{cases} \dfrac{dx}{dt} = \vartheta \cos\theta = r\dfrac{\omega_R + \omega_L}{2}\cos\theta, \\ \dfrac{dy}{dt} = \vartheta \sin\theta = r\dfrac{\omega_R + \omega_L}{2}\sin\theta, \\ \dfrac{d\theta}{dt} = \omega = r\dfrac{\omega_R - \omega_L}{L}. \end{cases} \quad (2.3)$$

Преобразуем систему (2.3). Получаем систему, для нахождения координат x, y и угла поворота робота в момент времени t:

$$\begin{cases} \dfrac{dx}{dt} = r\dfrac{\omega_R + \omega_L}{2}\cos\theta, \\ \dfrac{dy}{dt} = r\dfrac{\omega_R + \omega_L}{2}\sin\theta, \\ \dfrac{d\theta}{dt} = r\dfrac{\omega_R - \omega_L}{L}. \end{cases} \quad (2.4)$$

С начальными условиями:

$$\begin{cases} x(0) = x_0, \\ y(0) = y_0, \\ \theta(0) = \theta_0, \end{cases} \quad (2.5)$$

где x_0, y_0, θ_0 – начальное положение робота.

2.3 Понятие ПИД-регулятора. Описание системы управления мобильным роботом основанной на ПИД-регуляторе

Регулятор — устройство в системе управления, преобразующее ошибку регулирования в управляющее воздействие [11]. Полученное от регулятора управляющее воздействие применяется к системе, изменяя её состояние в нужную сторону.

В данной работе рассмотрим классические линейные регуляторы (пропорциональный, интегральный, пропорционально-интегральный и пропорционально-интегрально-дифференциальный).

П-регулятор вычисляет управляющее воздействие, пропорционально величине отклонения e (ошибки регулирования):

$$u_P = Kp \cdot e, \tag{2.6}$$

где Kp – пропорциональный коэффициент.

И-регулятор вычисляет управляющее воздействие, пропорционально интегралу от ошибки регулирования, уменьшая остаточную неравномерность регулирования:

$$u_I = \frac{1}{T_{\text{И}}} \int_0^{t_n} e\, dt, \tag{2.7}$$

где $T_{\text{И}}$ – время интегрирования, t_n – полное время регулирования.

ПИ-регулятор вычисляет управляющее воздействие, пропорционально сумме отклонения и интеграла отклонения регулируемой величины от заданного значения (комбинация формул (2.6) и (2.7)):

$$u_{PI} = Kp\left(e + \frac{1}{T_{\text{И}}} \int_0^{t_n} e\, dt\right).$$

ПИД-регулятор вычисляет управляющее воздействие, пропорционально отклонению, интегралу и скорости изменения отклонения регулируемой величины (дифференциальная составляющая помогает системе быстрее реагировать на изменения):

$$u_{PID} = Kp\left(e + \frac{1}{T_{\text{И}}} \int_0^{t_n} e\, dt + \text{Т}_{\text{упр}} \frac{de}{dt}\right), \tag{2.8}$$

где $\text{Т}_{\text{упр}}$ - время упреждения, характеризующее степень ввода производной в закон регулирования.

Введём следующую замену:

$$\begin{cases} K_{\text{I}} = Kp \dfrac{1}{T_{\text{И}}}, \\ K_{\text{D}} = Kp \cdot \text{Т}_{\text{упр}}. \end{cases} \tag{2.9}$$

Применяя к (2.8) замену (2.9), получим наиболее часто встречающийся вид закона регулирования:

$$u_{PID} = K_P \cdot e + K_I \cdot \int_0^{t_n} e\,dt + K_D \cdot \frac{de}{dt} \qquad (2.10)$$

ПИД-регулятор можно использовать для управления мобильным роботом при его движении по линии. Система, основанная на таком подходе, использует в качестве ошибки, значение, основанное на текущих показаниях датчиков света.

Рассмотренный выше робот, имеет три датчика света. Для простого движения по линии, без колебаний обычно достаточно два датчика. Выберем два передних датчика и рассмотрим реализацию движения по линии с их использованием. Ошибка движения в такой ситуации будет вычисляться как разность между показаниями левого и правого датчика:

$$e_1 = sensor_L - sensor_R \qquad (2.11)$$

где $sensor_L$ — показание левого датчика света;

$sensor_R$ — показание правого датчика света.

Такая ошибка принимает нулевое значение, в ситуации, когда левый и правый датчик выдают одинаковые значения (находятся по краям линии), изображено на рисунке 2.3.

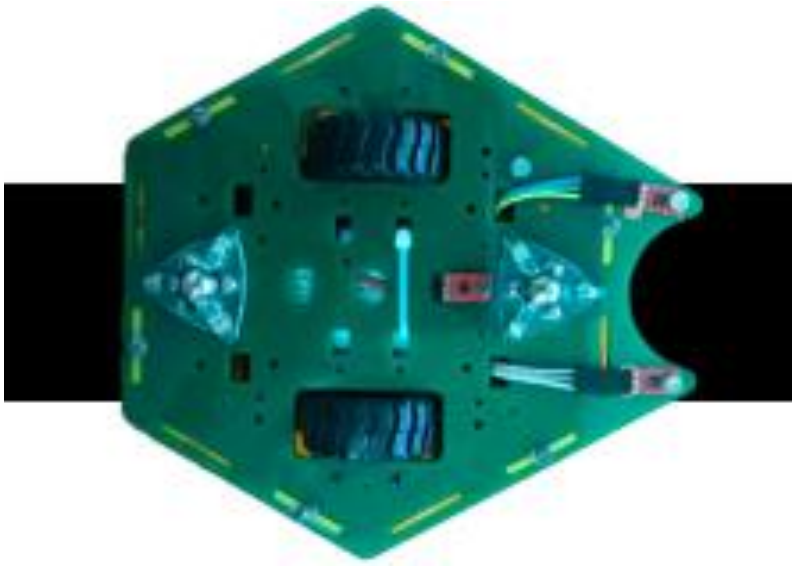

Рисунок 2.3 – Положение робота на линии, соответствующее нулевой ошибке

В случае нулевой ошибки управляющее воздействие также равно нулю (робот едет прямо).

Рассмотрим ситуации, когда $sensor_L < sensor_R$ и когда $sensor_L > sensor_R$.

В первой ситуации для корректировки движения робота, необходимо увеличить угловую скорость правого колеса и/или уменьшить угловую скорость левого колеса. Данная ситуация представлена на рисунке 2.4.

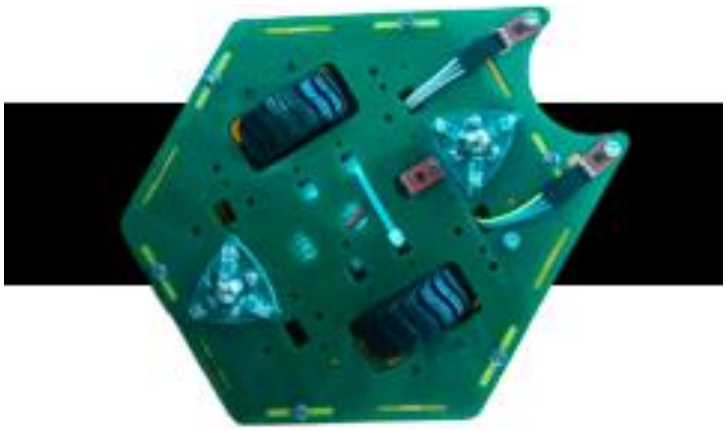

Рисунок 2.4 – Ситуация, при которой требуется корректировка угла поворота вправо

Вторая ситуация является противоположной, для того чтобы выровнять робота, требуется увеличить угловую скорость левого колеса и/или уменьшить угловую скорость для правого колеса. Данная ситуация представлена на рисунке 2.5.

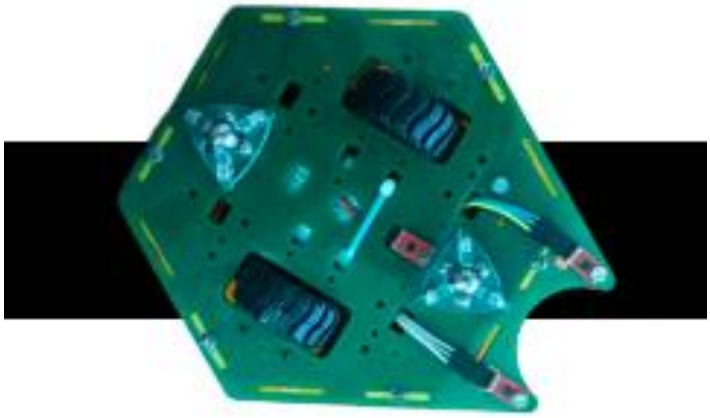

Рисунок 2.5 - Ситуация, при которой требуется корректировка угла поворота влево

Таким образом, высчитывая ошибку по формуле (2.11) и считая управляющее воздействие по формуле (2.10), изменение угловой скорости каждого колеса должно происходить следующим образом:

$$\begin{cases} \omega_R = \omega_{const} + u_1, \\ \omega_L = \omega_{const} - u_1, \end{cases} \quad (2.12)$$

где ω_{const} – угловая скорость, ожидаемая при нулевой ошибке.

Рассмотрим возможность использования третьего датчика и второго регулятора, для прямого ускорения робота (для данной цели хватит и П-регулятора).

Для этого рассмотрим вторую функцию ошибки, использующую показания центрального датчика:

$$e_2 = white - sensor_C, \tag{2.13}$$

где $white$ – значение датчика на белом;

$sensor_C$ – показание центрального датчика.

Управляющее воздействие u_2, в свою очередь, будет влиять на угловую скорость следующим образом:

$$\begin{cases} \omega_R = \omega_{const} + u_2, \\ \omega_L = \omega_{const} + u_2. \end{cases}$$

Комбинируя два регулятора, в процессе работы робот будет стремиться к белому цвету, используя ошибку, рассчитанную по формуле (2.13), в качестве входного сигнала для П-регулятора. ПИД-регулятор будет выполнять функцию выравнивания робота.

В случае поворота, если центральный датчик обнаружит белый цвет, П-регулятор будет получать нулевой управляющий сигнал, что позволит роботу выполнить поворот с помощью ПИД-регулятора.

Итоговые угловые скорости для левого и правого колеса:

$$\begin{cases} \omega_R = \omega_{const} + u_1 + u_2, \\ \omega_L = \omega_{const} - u_1 + u_2. \end{cases} \tag{2.14}$$

2.4 Нейроэволюция

Нейроэволюция – это семейство методов машинного обучения, которые используют эволюционные (генетические) алгоритмы для оптимизации параметров нейронных сетей. Конечным результатом нейроэволюции является оптимальная топология сети, обеспечивающая лучший показатель эффективности, по сравнению с начальным [3].

Генетический алгоритм представляет – это эвристический метод оптимизации, основанный на принципах теории эволюции и генетики. Он

применяется для нахождения оптимального решения в многовариантных пространствах поиска [4].

Процесс генетического алгоритма традиционно начинается с инициализации исходной популяции решений, которая может быть сформирована случайным образом или на основе предварительно заданных параметров [3]. Затем применяются операции скрещивания и мутации для генерации новых особей из родительских решений. Оценка новых особей производится на основании функции приспособленности, предназначенной для количественной оценки качества решения в контексте поставленной задачи. Отбор наиболее эффективных решений осуществляется для формирования новой популяции, и процесс повторяется на протяжении нескольких поколений до достижения заданного критерия остановки. Схема работы генетического алгоритма представлена на рисунке 2.6.

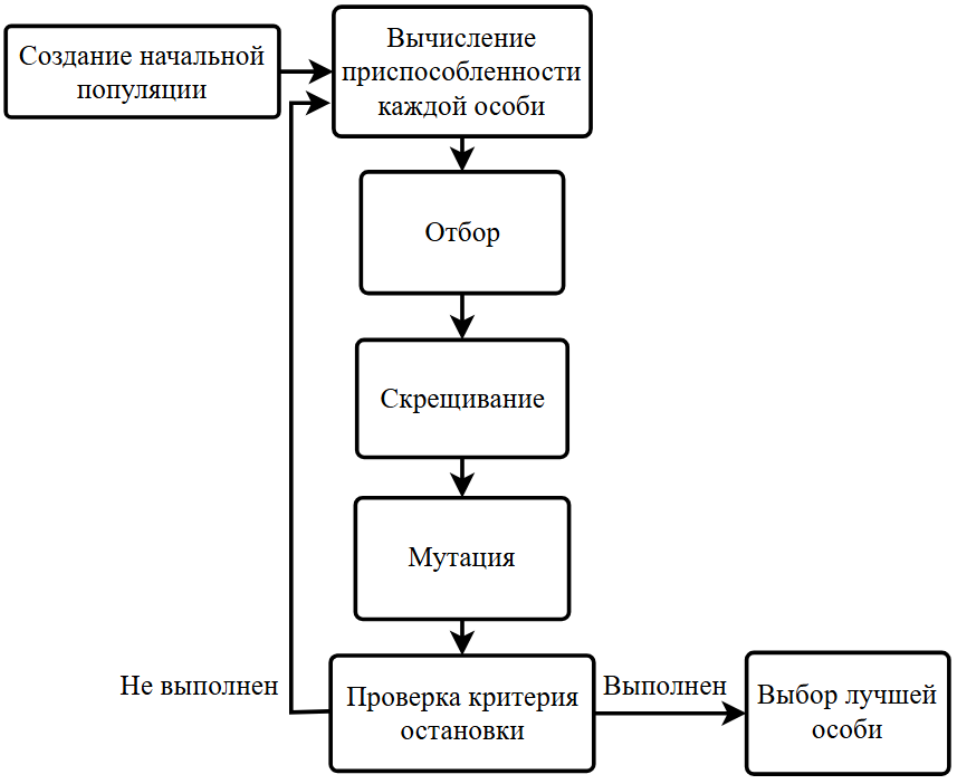

Рисунок 2.6 – Схема работы генетического алгоритма

Рассмотрим подробнее основные этапы работы генетического алгоритма, в качестве особи возьмём нейронную сеть, её параметры (веса, смещения и т. д.) являются её хромосомами. Например, массив, соответствующий весам связей между нейронами входного и скрытого слоя $W = [w_1, w_2, w_3, \dots, w_n]$.

Отбор представляет собой первую операцию, осуществляемую в отношении индивидов поколения. На этом этапе из всей совокупности особей

выбираются родительские особи для формирования следующего поколения. Однако не всегда лучшим методом отбора родительских особей является выбор исключительно лучших представителей популяции. Для обеспечения учета потенциальных особенностей всех особей в поколении необходимо интегрировать в последующие этапы репродуктивного процесса как лучших, так и наименее успешных особей.

Таким образом в процессе отбора осуществляется селекция определенного числа особей, которые затем переходят к последующим этапам генетического алгоритма. Схема отбора представлена на рисунке 2.7.

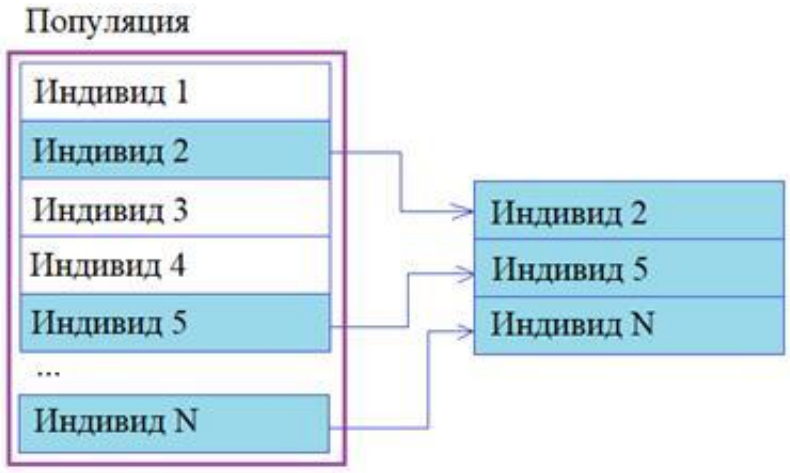

Рисунок 2.7 – Схема операции отбора

Второй операцией генетического алгоритма является скрещивание (кроссинговер). На данном этапе параметры родительских особей комбинируются с использованием определенных алгоритмов для формирования особей нового поколения [4]. В большинстве случаев родительские особи разбиваются на наборы генетических параметров, которые затем обмениваются между собой. Схема скрещивания представлена на рисунке 2.8.

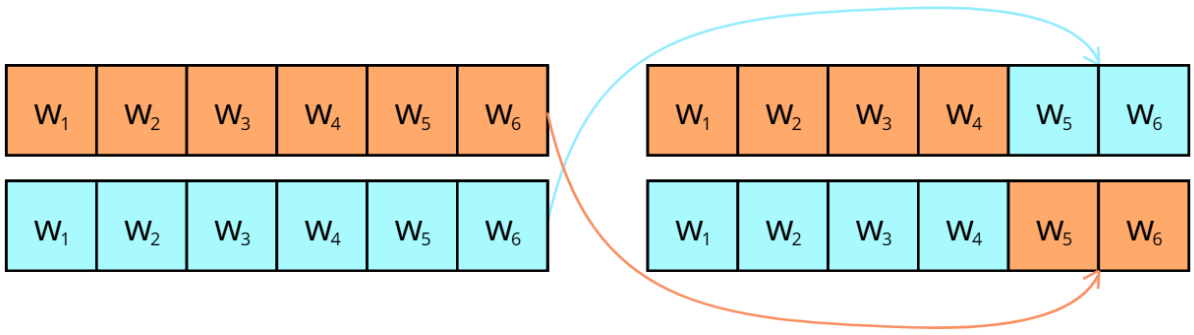

Рисунок 2.8 – Схема операции скрещивания

В контексте обработки нейронных сетей на рисунке 2.8 представлена схема скрещивания весовых коэффициентов нейросетей родительских особей, при которой случайным образом выбирается точка раздела, после чего происходит обмен соответствующими сегментами. В результате образуются две новые особи, а родительские особи, не участвующие в скрещивании, сохраняются. Таким образом, размер популяции после данной операции не изменяется.

Последней операцией называется мутация. Она выполняется над сформированной популяцией и, с определенной вероятностью, случайным образом модифицирует значения отдельных параметров индивидуумов. Схема мутации представлена на рисунке 2.9.

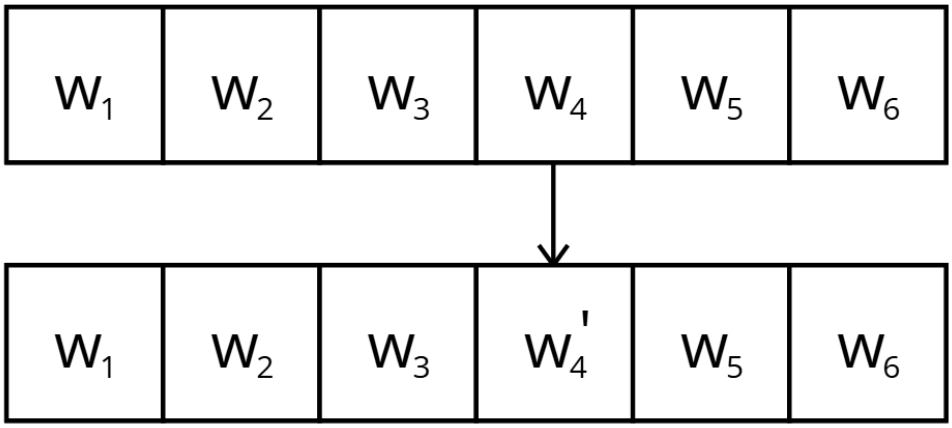

Рисунок 2.9 – Схема операции мутации

Мутация обеспечивает расширение поискового пространства для решения задачи и поддержание генетического разнообразия в популяции. В результате мутации особь может приобрести новое свойство, которое повысит её конкурентоспособность в рамках популяции. В перспективе это увеличивает вероятность успешного размножения и закрепления полезного признака. Таким образом, посредством мутационного процесса осуществляется оптимизация решения.

2.5 Нейроэволюционная система управления роботом

В данной системе управления решения по управлению движением робота, будет принимать заранее обученная нейронная сеть. Нейронную сеть будем обучать с помощью нейроэволюции, а именно упрощённой версией алгоритма NeuroEvolution of Augmenting Topologies, в которой не будут изменяться связи между нейронами. Опишем работу данного алгоритма [2].

Реализация алгоритма будет проводится средствами языка Python, поэтому для начала рассмотрим упрощённую кинематику робота. Рассмотрим процесс кинематический закон движения робота с дискретным временем.

Пусть в начале каждого дискрета времени позиция роботов соответствует точке с координатами О (x_{i-1}, y_{i-1}), а размер дискрета времени $\Delta t = 1c$. Один пиксель программной визуализации выберем равным 5мм.

Тогда кинематический закон движения будет иметь следующий вид [7]:

$$\begin{cases} x_i = x_{i-1} + \vartheta_{x,i-1} \cdot \Delta t + a_{x,i-1} \cdot \dfrac{\Delta t^2}{2}, \\ y_i = y_{i-1} + \vartheta_{y,i-1} \cdot \Delta t + a_{y,i-1} \cdot \dfrac{\Delta t^2}{2}, \end{cases}$$

где $\vartheta_{x,i-1}$ и $\vartheta_{y,i-1}$ — проекции вектора скорости $\vec{\vartheta}_{i-1}$ в i -1 дискрет времени, на оси Ох, Оу;

$a_{x,i-1}$ и $a_{y,i-1}$ — проекции вектора ускорения $\vec{a}_{i-1}$.

Перемещение r_i за один дискрет времени можно найти как:

$$|\vec{r_i}| = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}.$$

Таким образом, зная угол поворота робота φ_{i-1} в конкретный дискрет времени, мы можем найти изменение координат робота, как:

$$\begin{cases} \Delta x_{i-1} = \vec{r_i} \cdot \sin \varphi_{i-1}, \\ \Delta y_{i-1} = \vec{r_i} \cdot \cos \varphi_{i-1}. \end{cases} \quad (2.15)$$

Пусть T_j – время, затраченное j-ым роботом для достижения финиша:

$$T_j = \Delta t \cdot n_j,$$

где n_j – количество дискретов времени, в которое робот совершал движение.

Тогда, весь путь, пройденный j-ым роботом, можно найти как:

$$s_j = \sum_{i=1}^{n\_j} |\overrightarrow{r_{i,j}}|. \quad (2.16)$$

В качестве модели робота, был выбран круг (радиусом 20 пикселей) с пятью радарами: 0, 45, 90, -45 и -90 градусов, идущими из центра. Модель робота изображена на рисунке 2.10.

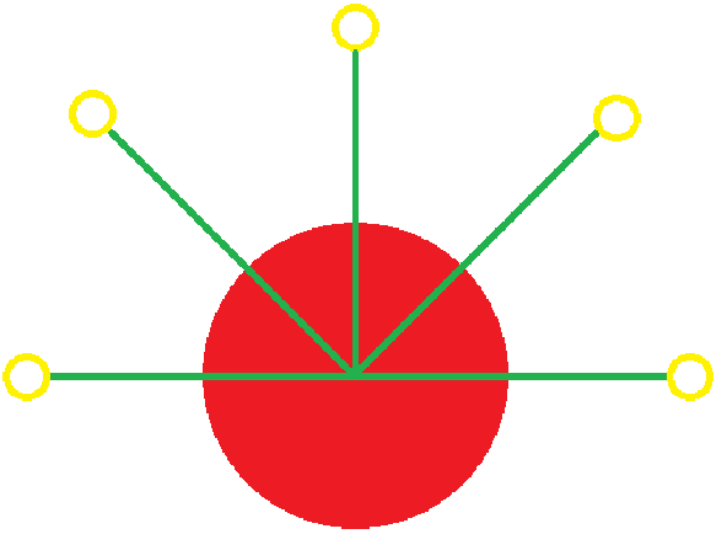

Рисунок 2.10 — Модель робота

Каждому роботу в соответствие поставим свою нейронную сеть прямого распространения. С шестью входными нейронами, пятью нейронами на скрытом слое и двумя нейронами на выходном слое. Схема данной нейронной сети представлена на рисунке 2.11.

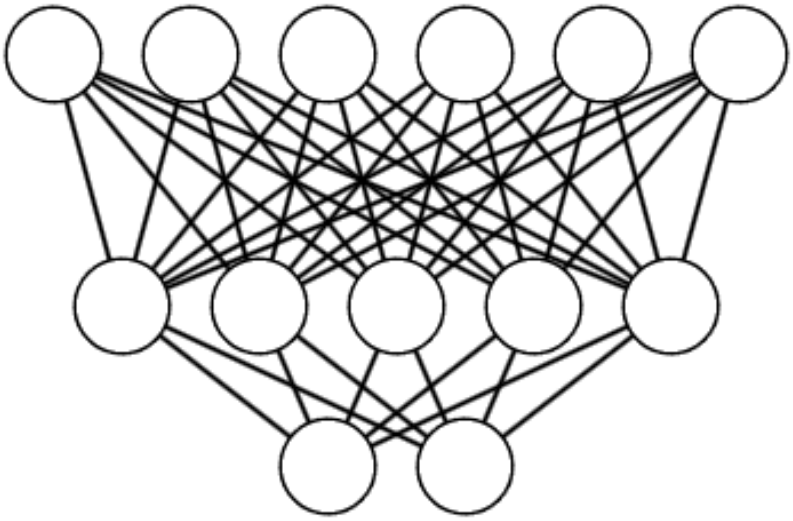

Рисунок 2.11 – Схема нейронной сети

Входная информация. В нашем случае будет рассматриваться массив, состоящий из пяти элементов $input_{data_i}$ (i-тому нейрону на вход подаётся $input_i$ элемент входного массива).

$$input_{data} = [r_0, r_1, r_2, r_3, r_4, \vartheta_i],$$

где r_i – показание с i-го радара;

ϑ_i — текущая скорость робота.

Выходная информация. В нашем случае мы имеем два нейрона, который, в результате расчётов, будут давать нам значения output_data, в диапазоне [-1,1].

Связи между нейронами имеют свои веса, пусть w_0 и w_1 массивы весов между входным и скрытым, и скрытым и выходным слоями соответственно.

Каждый нейрон имеет несколько параметров:

1. activation — функция активации. В нашем случае для получения значений в нужном диапазоне был выбран тангенс гиперболический, график которого представлен на рисунке 2.12.

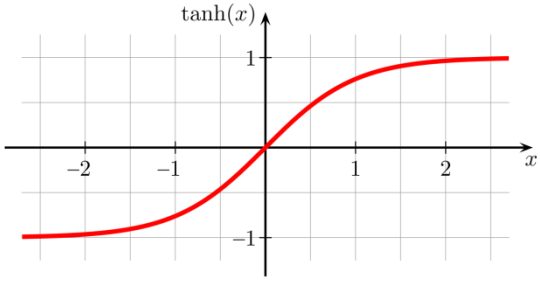

Рисунок 2.12 – График функции гиперболического тангенса

2. aggregation – функция обрабатывающая входные значения нейрона, в нашем случае была выбрана функция mean (среднее арифметическое).

3. bias (смещения) - смещения нужны для того, чтобы иметь возможность получать выходной результат, путем сдвига графика функции активации вправо или влево.

4. response – значение используемое в вычислениях в библиотеке NEAT-Python (библиотека далее используемая для программной реализации).

Изменение данных параметров, будет производится по средствам работы генетического алгоритма, который, будет рассмотрен далее.

Рассмотрим начальные значения bias, response, w_1 , w_2 . Начальные значения задаются случайным образом, по нормальному (гауссовскому) закону распределения вероятности N[0,1] (коэффициент сдвига $\mu = 0$ и единичной дисперсией $\sigma^2 = 1$) [21].

Последним шагом станет рассмотрение работы генетического алгоритма, по средствам которого будет проводится обучение нейронных сетей. Как ранее упоминалось, для программной реализации будет использоваться библиотека NEAT-Python, поэтому будем рассматривать генетический алгоритм, реализованный в ней.

Введём следующие понятия в контексте нейроэволюции:

Ген — параметр нейросети для определенного элемента (узла или соединения) [21].

Геном — набор генов, кодирующих фенотип (нейросеть) [21].

Гомологичный ген – ген, произошедший от общего предка [21].

Избыточный ген – ген, не произошедший от общего предка [21].

В текущей версии NEAT-Python используется популяционный подход с геномами для создания искусственных нейронных сетей. Геномы содержат:

1. Гены узлов для структуры нейронов.
2. Гены связей для соединений между нейронами.

Качество генома оценивается функцией соответствия, генерирующей числовой показатель успешности решения задачи. Эволюция происходит в течение заданного числа поколений через отбор и мутации наиболее приспособленных особей.

В нашем случае в качестве результата функции соответствия будем использовать следующую функцию для i-ого робота (функцию наград):

$$\text{fitness}_i = \begin{cases} s_i + \dfrac{s_i}{n_i}, \text{если робот достиг финиша,} \\ s_i, \text{иначе,} \end{cases} \tag{2.17}$$

где s_i — весь путь, пройденный i-ым роботом, вычисляемый по формуле (2.16);

n_i — количество дискретов времени, в которое робот совершал движение.

Изменение функции наград, для роботом достигших финиша, сделано для того, чтобы роботы стремились достичь финиша за наименьшее число дискретов времени.

Далее рассмотрим процесс создания новой популяции, после прохождения трассы предыдущим поколение, он включает в себя 3 этапа: селекция, размножение и мутация.

При селекции выбирается 5 роботов с наибольшими fitness_i, которые на этапах размножения и мутации создадут новое поколение. Далее полученные особи переходят к этапам скрещивания и мутации. Для мутации так же выберем следующие параметры:

1. Вероятность мутации: 10%. Это значит, что любой параметр родителя может быть изменён (как в меньшую, так и в большую сторону) на величину равную силе мутации, при создании потомка.
2. Сила мутации: 0.5.

Введём следующие понятия:

1. Смерь робота – ситуация, когда робот касается границы трассы и больше не участвует в заезде (робот мертв).
2. Робот жив – ситуация, когда робот находится на трассе и может продолжать движение, то есть робот не мертв.

3. Финиширование робота – ситуация, когда робот достигает конечной зоны трассы (финиша).

4. Информирование нейросети – передача информации input_data с радаров на входной слой нейросети, для дальнейшей её работы.

5. Действие робота – изменение угла поворота робота на величину $\Delta\varphi = 10° \cdot output_0$, и скорости на величину равную $5\text{пк} \cdot output_1$.

6. Награды робота – значения fitness_i , которые передаются генетическому алгоритму, для выбора лучших геномов популяции.

Итак, опишем алгоритм обучения целиком:

1. Инициализация популяции. Создаём 30 роботов и устанавливаем их в точку старта.

2. Даём в соответствие каждому роботу свою нейронную сеть.

3. Проверяем есть ли живые роботы, если нет, то переходим к шагу №7.

4. Производим информирование нейросетей, соответствующих живым роботам.

5. Для каждого робота получаем $output_i$. Производим действия роботов.

6. Возвращаемся к шагу №3.

7. Раздаём награды каждому роботу, в зависимости, от расстояния на которое они проехали.

8. Работа генетического алгоритма. На основании полученных наград, происходит создание новой популяции из 30 геномов (нейросетей), которые заменят старые.

9. Если критерий остановки выполнен, то завершаем работу, иначе переходим к шагу №1. Критерий остановки в следующих пунктах будет выбираться отдельно, для достижения необходимых результатов.

Далее лучшую нейронную сеть можно импортировать в MATLAB Simulink и использовать для управления цифровым двойником на различных трассах.

2.6 Выводы по второй главе

В начале данной главы была рассмотрена физическая модель тестового мобильного робота RoboCake. Полученные данные послужили основой для последующего теоретического анализа, позволившего выделить ключевые параметры, оказывающие влияние на динамику движения и формирование управляющих сигналов.

Далее была построена математическая модель движения робота. Моделирование обеспечило формализацию процессов перемещения, что создало основу для реализации этих принципов в цифровом двойнике.

Во второй половине главы были рассмотрены ПИД-регуляторы, как базовые элементы классических систем управления. На основании проведенного анализа были описаны две системы управления, и сделан вывод о высокой эффективности ПИД-регуляторов для задач следования по линии и стабилизации движения в условиях слабо изменяющейся среды.

В конце главы был рассмотрен нейроэволюционный подход, интегрирующий методы нейронных сетей и эволюционные алгоритмы. На основании данного подхода был описан процесс обучения нейронной сети, использование которой позволяет построить ещё одну систему управления. Данная система обладает высокой адаптивностью к различным трассам, что делает её особенно перспективной для применения в сложных условиях.

Таким образом, во второй главе был сформирован теоретический базис, необходимый для реализации и оценки различных систем управления мобильным роботом. Полученные модели и алгоритмы будут служить основой для практической части исследования, представленной в следующей главе, где планируется реализация и экспериментальная проверка рассмотренных подходов.

ГЛАВА 3
РЕАЛИЗАЦИЯ ЦИФРОВОГО ДВОЙНИКА И СИСТЕМ УПРАВЛЕНИЯ МОБИЛЬНОГО РОБОТА

3.1 Моделирование площадки и цифрового двойника

В качестве площадки для испытаний робота необходимо создать квадратную поверхность размерами 5м×5м. Так же по краям площадки необходимо создать стенки, чтобы робот не мог выехать за её пределы.

MATLAB Simulink позволяет работать с 3d объектами: создавать новые или использовать готовые, задавать их параметры (гравитация, центры масс, инерция и др.) и размещать их относительно центра мира или других объектов.

Для начала были созданы и соединены блоки World Frame, Solver Configuration и Mechanism Configuration:

- World Frame создаёт оси мира (X, Y, Z);
- Solver Configuration настраивает численный решатель, для решения физических уравнений движения и взаимодействия моделей;
- Mechanism Configuration задаёт глобальные параметры модели.

Расположение данных блоков представлено на рисунке 3.1.

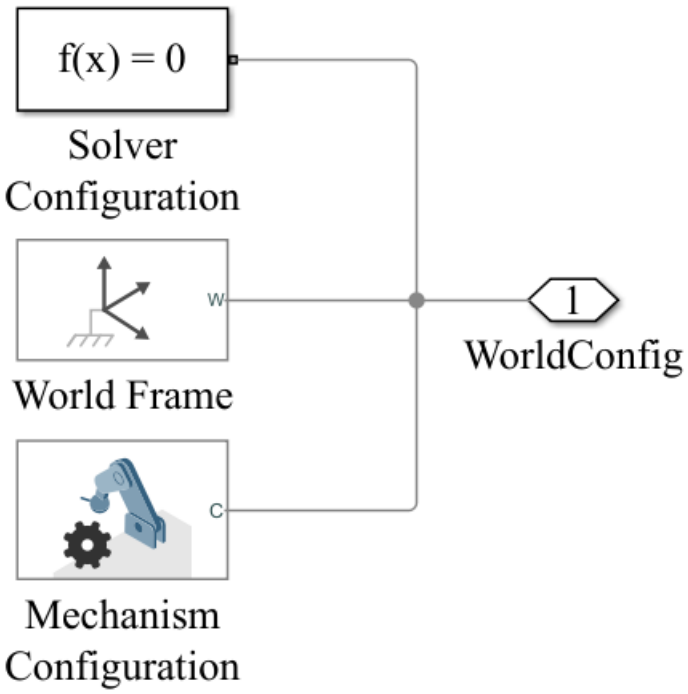

Рисунок 3.1 – Подсистема, задающая конфигурацию мира

Для создания платформы с стенками создаём 5 блоков Brick Solid, настраиваем их размеры подключаем к порту подсистемы с конфигурацией мира. Размещение стен и платформы относительно осей мира, осуществляется с помощью блоков Rigid Transform. Так же добавляем блоки Spatial.

Contact Force, которые соединяем с блоками стен/площадки и портом, который будет соединятся с роботом, для задания их взаимодействия. Если этого не сделать, то робот будет проваливаться сквозь карту. Вид подсистем для создания стен и взаимодействия их с роботом представлен на рисунках 3.2 и 3.3.

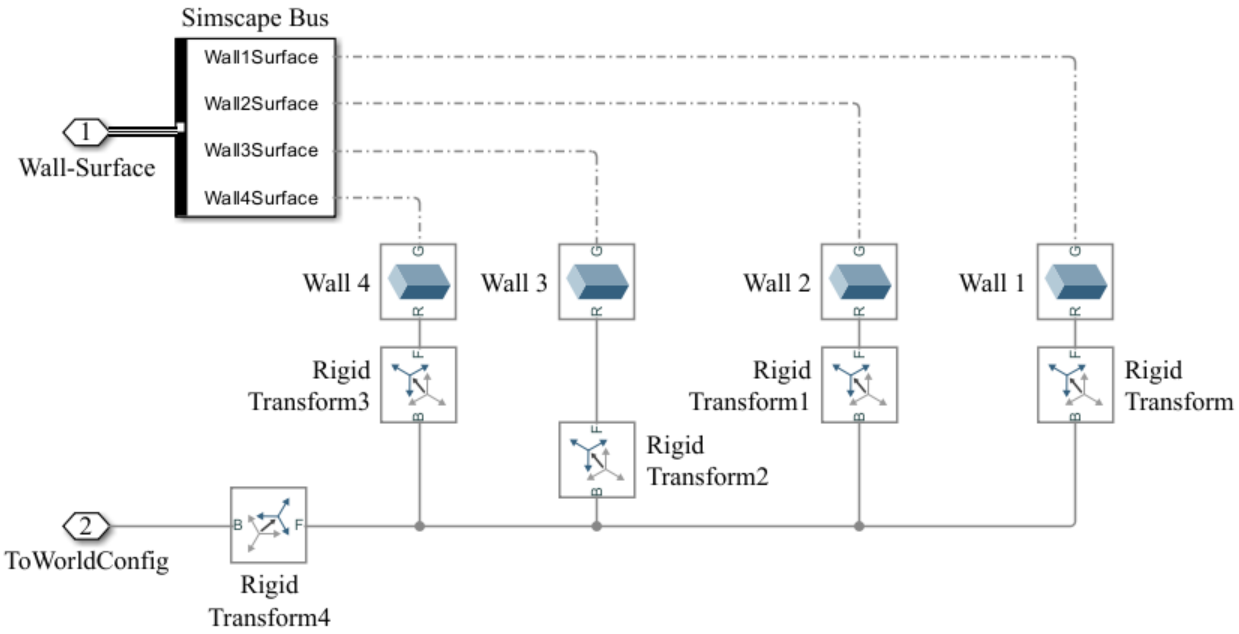

Рисунок 3.2 – Подсистема с 3d моделью стен

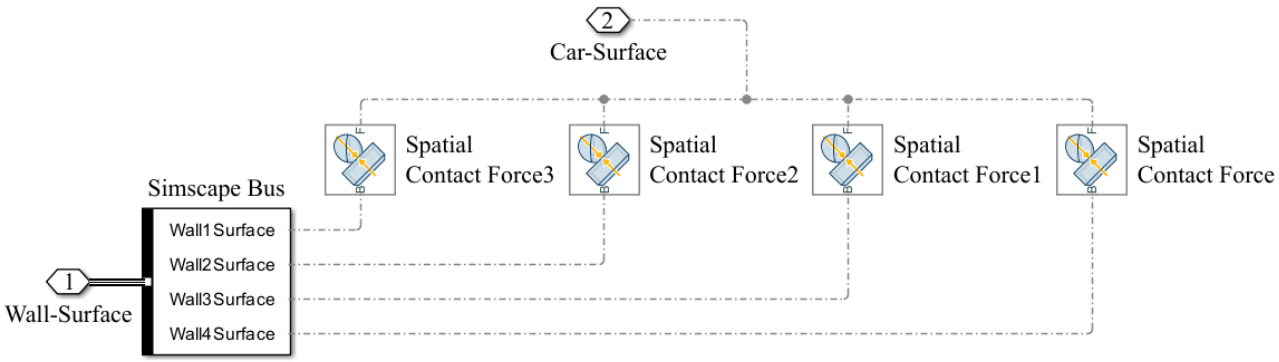

Рисунок 3.3 – Подсистема, задающая взаимодействие робота с стенами

Получаем следующую площадку для испытаний робота, представленную на рисунке 3.4.

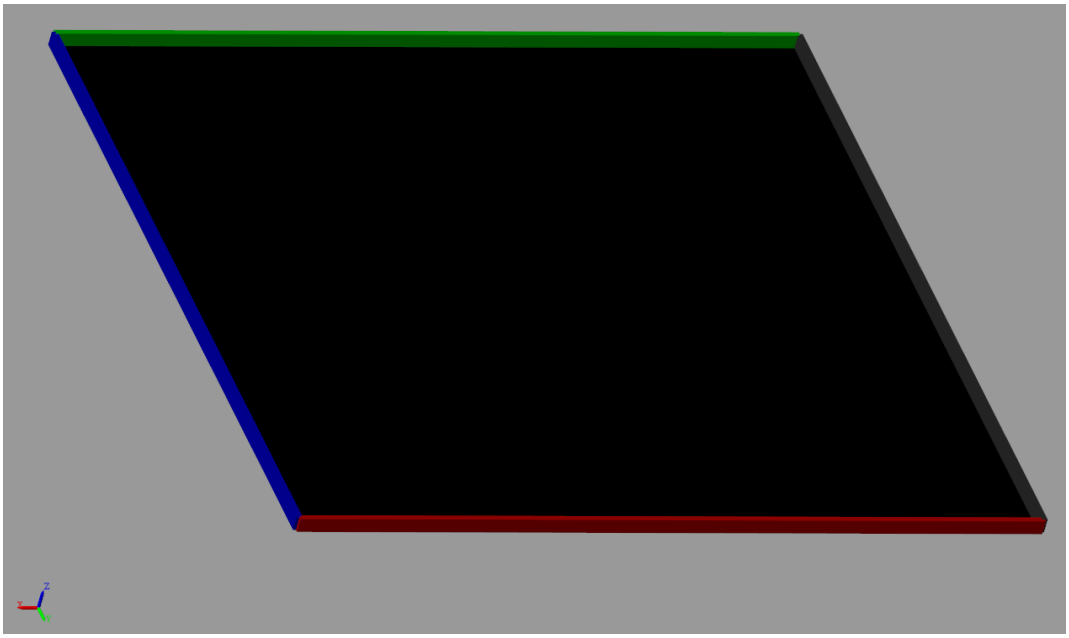

Рисунок 3.4 – Площадка для испытаний робота

Для создания робота, необходимо создать его платформу, колёса, стабилизаторы и датчики. Платформа была создана как отдельная 3d модель, поэтому подключаем её через блок File Solid. Датчики создадим в той же подсистеме. В MATLAB Simulink нет готового решения для датчиков света, поэтому создаём их с помощью Brick Solid.

Текущие координаты датчика, будем получать используя блок Transform Sensor. Он позволяет отслеживать изменения осей 3d объекта относительно других осей (осей мира). В настройках Transform Sensor указываем что хотим отслеживать перемещение вдоль оси X и оси Y. Подключаем блок Transform Sensor с одной стороны к фрейму (осям) датчика, с другой стороны к фрейму мира. Transform Sensor выдаёт значения перемещения физическими сигналами (в метрах), но нам удобнее перевести их в числовые сигналы, для этого проводим их через блок PS-Simulink Converter. Проделываем тоже самое для остальных двух сенсоров, объединяем все выходные сигналы через блок Mux. И выводим через порт с названием Sensors_cordinates из подсистемы робота. Получаем подсистему для получения координат, представленную на рисунке 3.5.

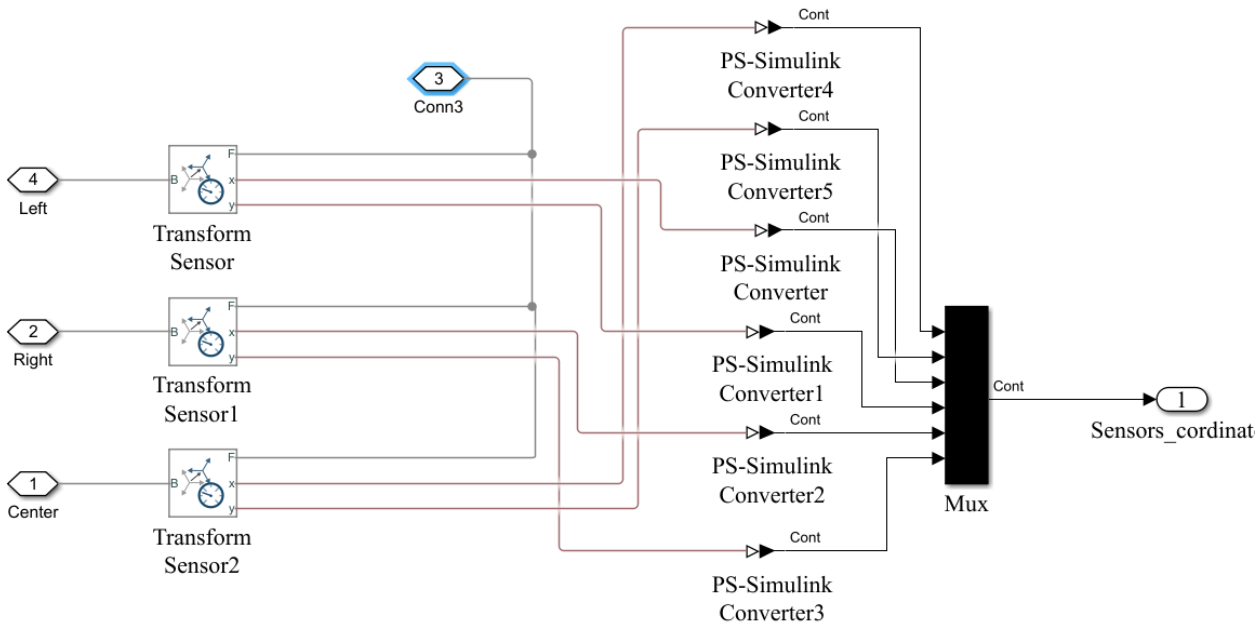

Рисунок 3.5 – Подсистема получения координат сенсоров

Соединяем подсистему для получения координат с платформой и сенсорами, получаем подсистему, представленную на рисунке N.

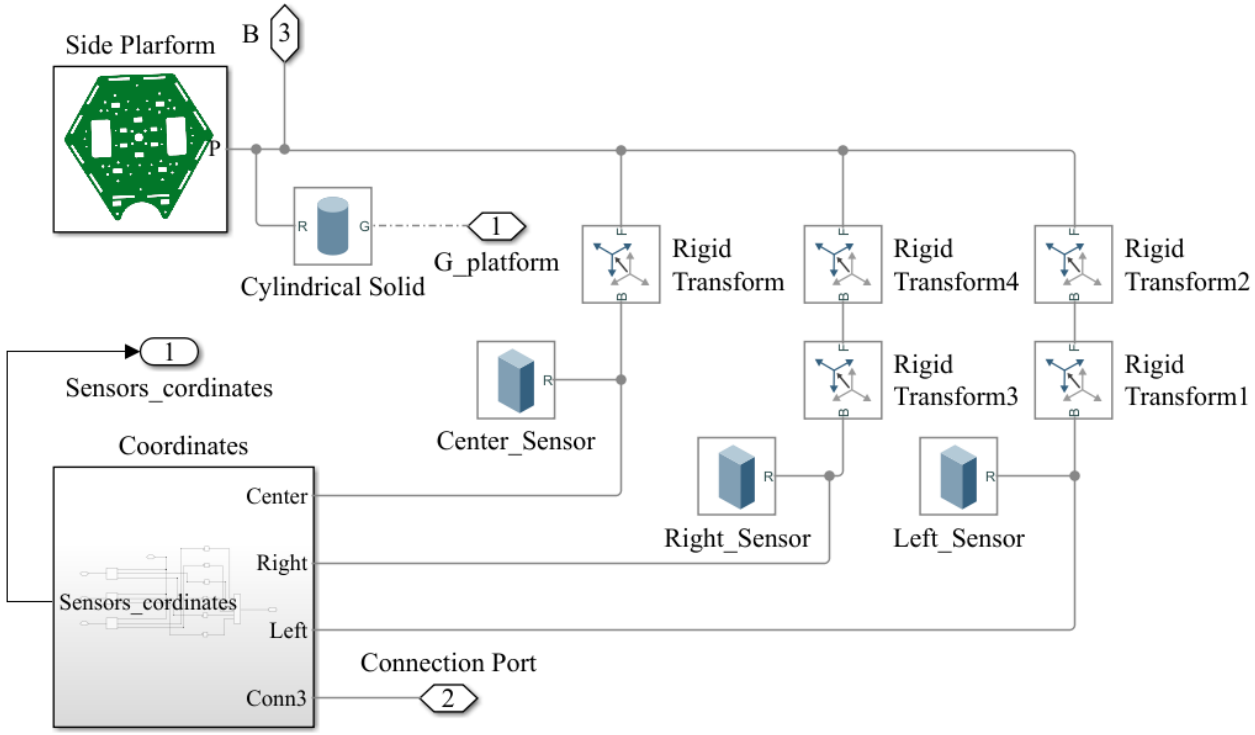

Рисунок 3.6 – Подсистема платформы и сенсоров

Колёса создаём с помощью Cylindrical Solid. Их необходимо соединить с Revolute Joint для добавления им возможности вращаться. Блок Revolute Joint

имеет порт для подачи угловой скорости. Получаем подсистему для одного колеса, представленную на рисунке 3.7.

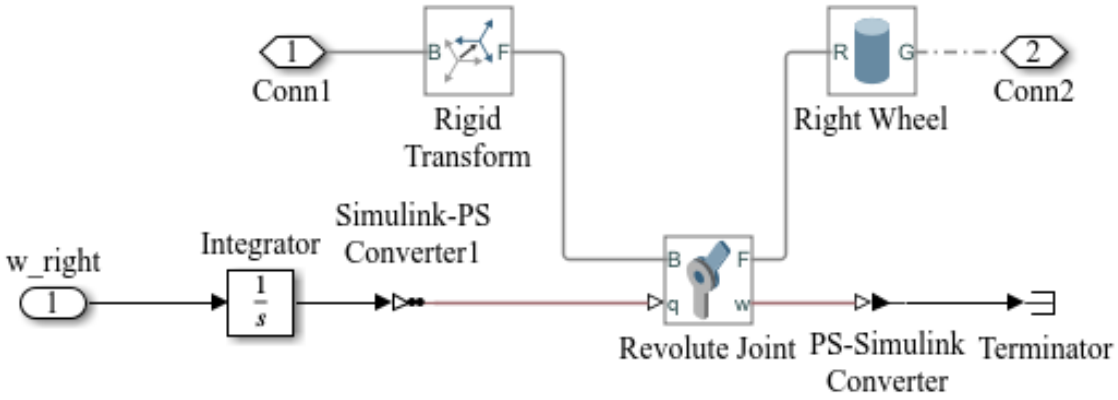

Рисунок 3.7 – Подсистема колеса

Стабилизаторы создадим с помощью Spherical Solid, они не должны вращаться, поэтому Revolute Joint не требуется. Подсистема для одного стабилизатора представлена на рисунке 3.8.

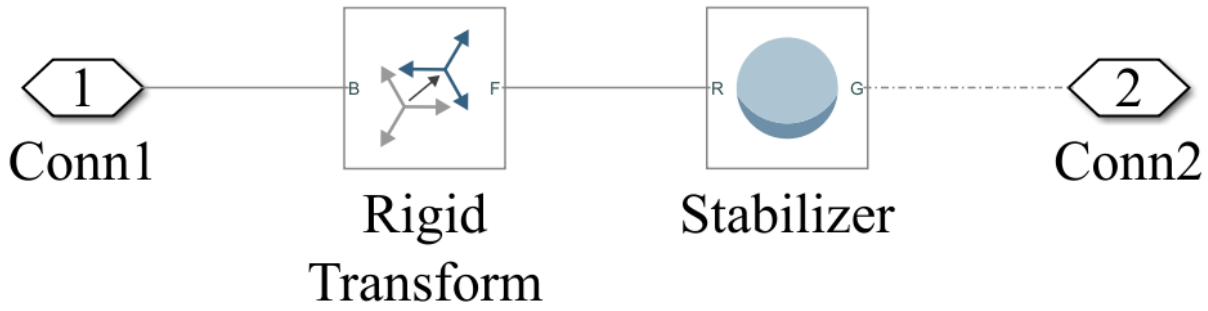

Рисунок 3.8 – Подсистема стабилизатора

Таким образом подсистема с моделью робота имеет следующий вид, представленный на рисунке 3.9.

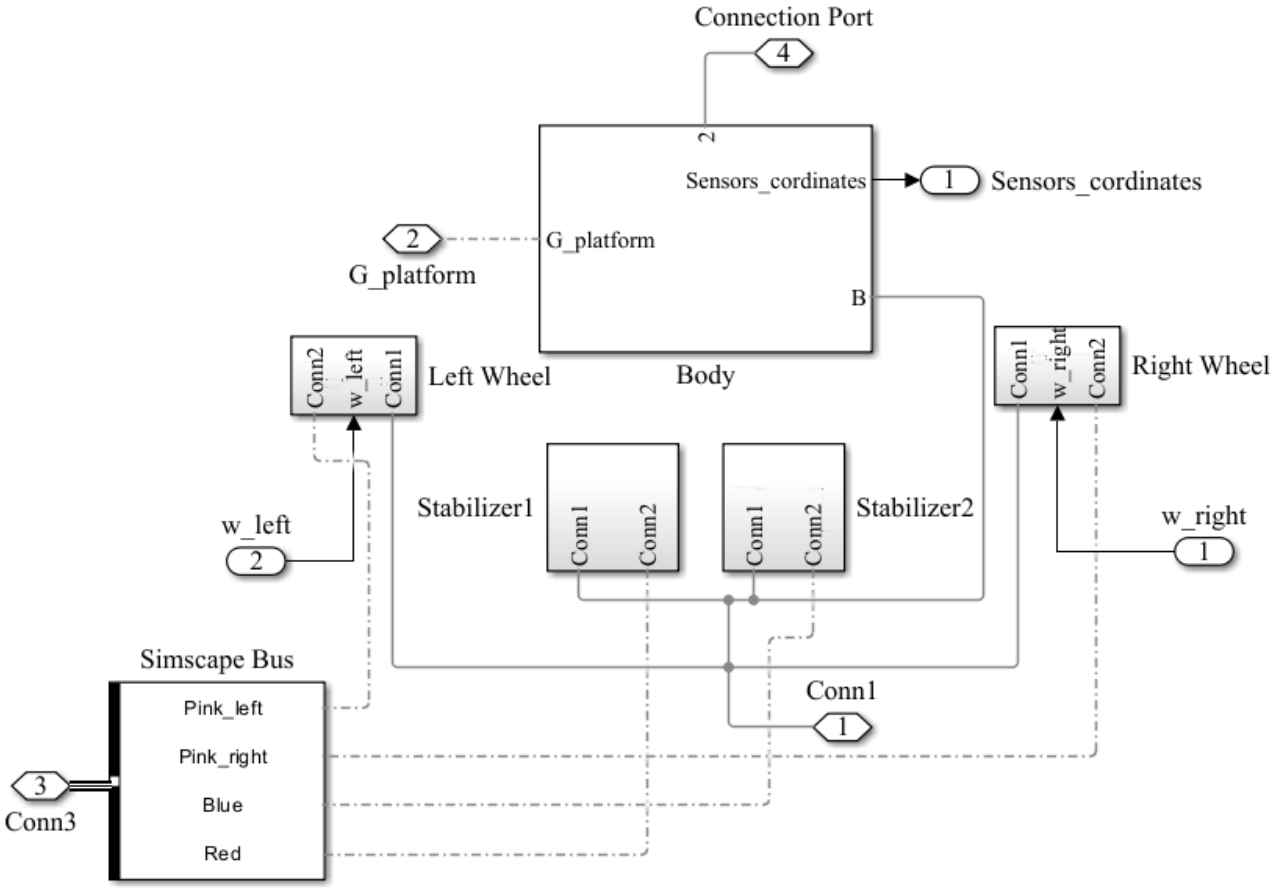

Рисунок 3.9 – Вид подсистемы с моделью робота

Важным шагом является соединение колёс и стабилизатором с платформой через Spatial Contact Force, для обеспечения их взаимодействия, как было сделано с стенами. Также необходимо соединить их с Planar Joint и Prismatic Joint, что позволит дать им две степени свободы перемещения по осям X и Y, и одну степень свободы для вращения по оси Z (здесь имеются ввиду собственные оси объектов колёс и стабилизаторов). Внешний вид модели робота представлен на рисунке 3.10.

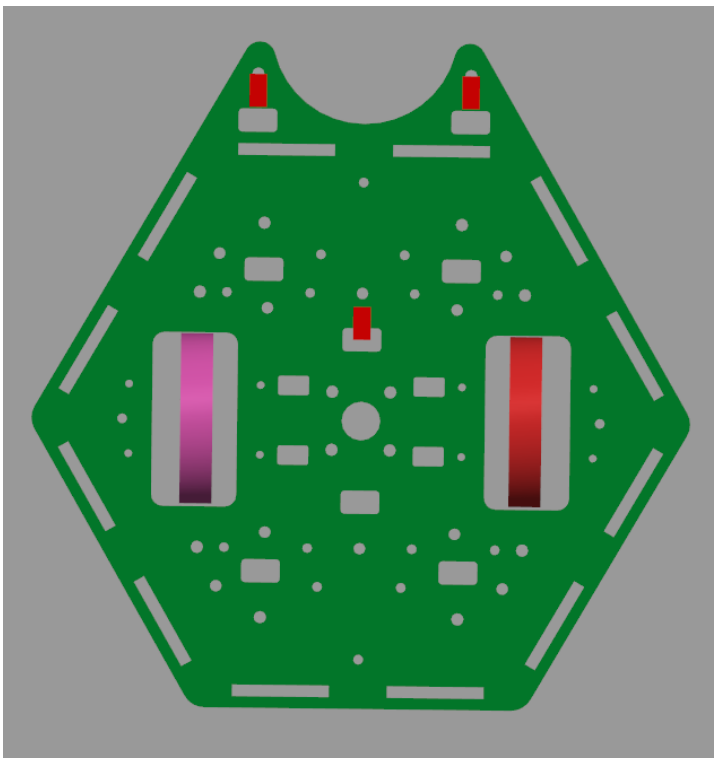

Рисунок 3.10 – Модель робота

3.2 Реализация ручного управления

Как было сказано выше, движение модели робота осуществляется путём передачи значений угловой скорости в блок Revolute Joint каждого колеса.

При ручном управлении оператор задаёт угловую скорость вручную. В качестве интерфейса для ручного управления создадим два блока Knob. Создадим два блока Constant, выдающих константный сигнал и соединим их с блоками Knob для возможности задания констант через тумблеры. Зададим тумблерам полученный ранее диапазон скоростей. Получаем следующую схему для ручного управления, представленную на рисунке 3.11.

Ввиду того что блок Knob обновляет значение связанного с ним блока Constant, полученная схема ручного управления позволяет осуществлять управление роботом в режиме реального времени.

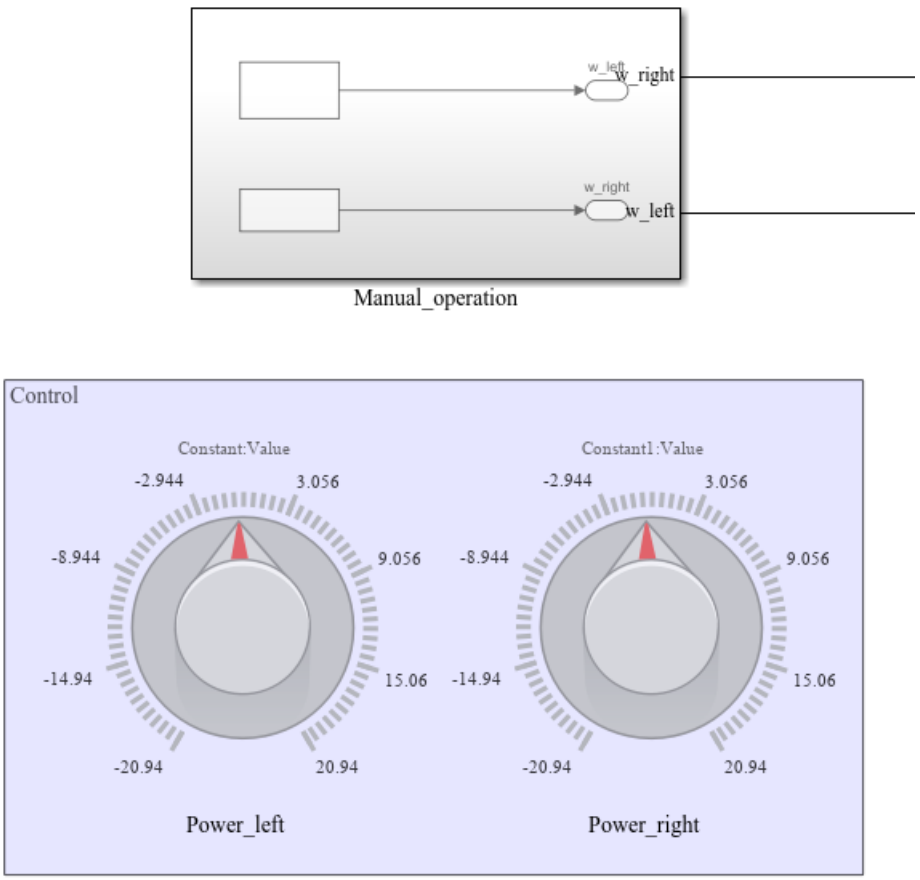

Рисунок 3.11 – Схема ручного управления

3.3 Реализация системы управления основанной на ПИД-регуляторах

В связи с тем, что MATLAB Simulink не имеет готовых датчиков света, необходимо реализовать аналоговые датчики.

Поверхность, как было сказано выше, имеет размеры 5м × 5м. Поставим ей в соответствие изображение с чёрной линией на белом фоне, размерами 1000px × 1000px. Таким образом 1px изображения будет соответствовать 5мм площадки.

Расстояние между передними датчиками составляет 5,5 см. Выберем ширину линии равной 5,5см ≈ 11px.

Представляя изображение как матрицу, будем считать показания датчика, как среднее арифметической элементов её минора, размера 3×3 с центром в элементе, соответствующем текущим координатам датчика.

Создадим подсистему для расчёта показаний датчиков света. В ней через входной порт получаем координаты датчиков, распаковываем их через Demux.

Далее создаём блок MATLAB Function, реализующий функцию, для расчёта показаний датчиков. Код данной функции представлен на рисунке 3.12.

```
function [sensor_l,sensor_r,sensor_c]  = fcn(x_l,y_l,x_r,y_r,x_c,y_c,image_matrix,meters_in_px)

[rows, cols] = size(image_matrix);

x_l_px = rows - round(x_l/meters_in_px + rows/2);
y_l_px = round(y_l/meters_in_px + rows/2);

x_r_px = rows - round(x_r/meters_in_px + rows/2);
y_r_px = round(y_r/meters_in_px + rows/2);

x_c_px = rows - round(x_c/meters_in_px + rows/2);
y_c_px = round(y_c/meters_in_px + rows/2);

minor_l = image_matrix(x_l_px-1:x_l_px+1, y_l_px-1:y_l_px+1);
minor_r = image_matrix(x_r_px-1:x_r_px+1, y_r_px-1:y_r_px+1);
minor_c = image_matrix(x_c_px-1:x_c_px+1, y_c_px-1:y_c_px+1);

sensor_l = sum(minor_l,'all')/9;
sensor_r = sum(minor_r,'all')/9;
sensor_c = sum(minor_c,'all')/9;

end
```

Рисунок 3.12 – Код функции, считающей показания датчиков

Далее выводим показания датчиков через выходные порты. Получаем вид подсистемы, представленный на рисунке 3.13.

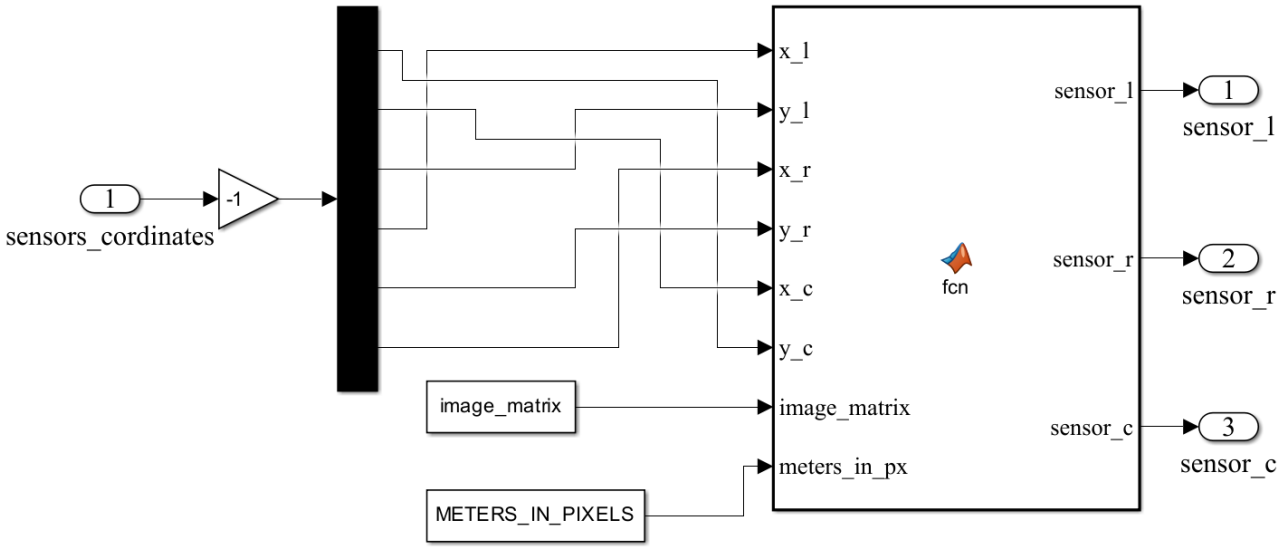

Рисунок 3.13 – Подсистема расчёта показаний датчиков

Далее необходимо реализовать расчёт ω_R и ω_L по формулам (2.12) и (2.14). Для этого через входные порты получаем показания сенсоров и подключаем их к блоку MATLAB Function, реализующему функцию, считающую значение ошибки по формулам (2.11), (2.13).

Для системы, работающей по формуле (2.12), передаём ошибку в блок PID Controller, который возвращает управляющее воздействие. Управляющее воздействие через блок Sum складываем с $\omega_{const} = 10$ рад/c, полученным из блока Constant. К полученному сигналу суммы прибавляем/отнимаем управляющее воздействие ПИД-регулятора, получая реализацию формулы (2.12). И выводим полученные угловые скорости через выходные порты. Получаем схему, представленную на рисунке 3.14.

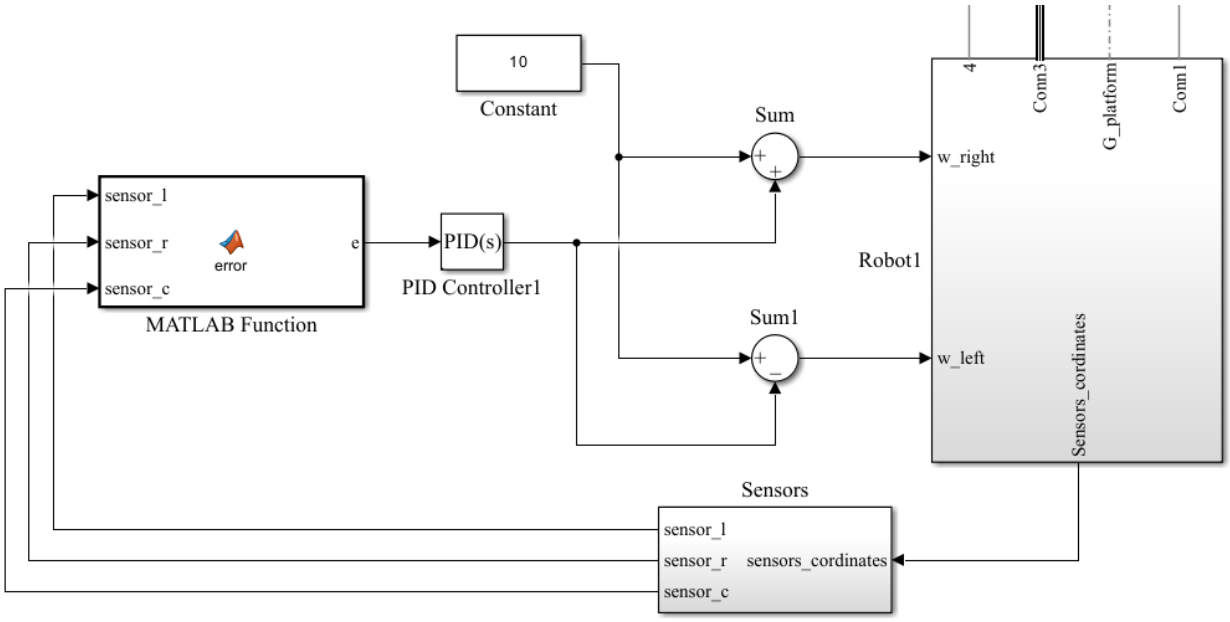

Рисунок 3.14 – Схема управления на ПИД-регуляторе

Аналогично получаем систему, реализующую формулу (2.14). Вид данной системы представлен на рисунке 3.15.

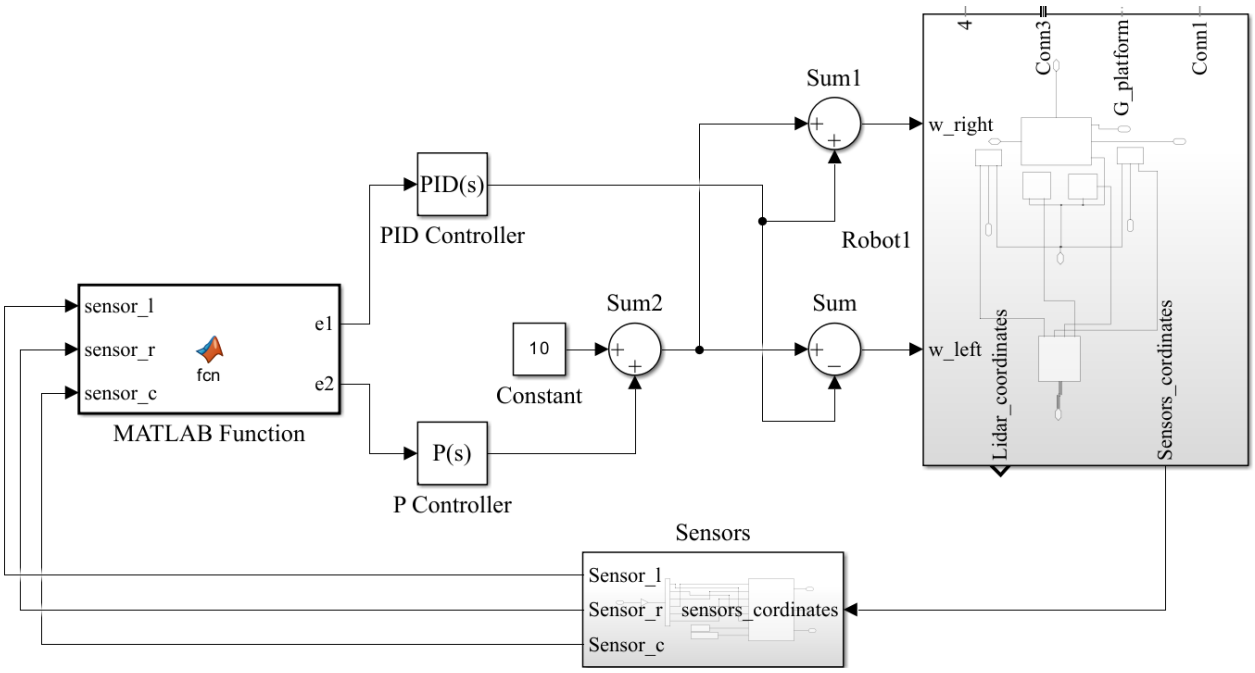

Рисунок 3.15 – Схема управления на ПИД-регуляторе и П-регуляторе

3.4 Настройка регуляторов

Важным этапом в работе с линейными регуляторами, является этап их настройки, с целью получения желаемых результатов. В частности, для настройки ПИД-регуляторов, стоит задача подбора их коэффициентов из формулы (2.10): K_P, K_I, K_D. Данный этап является самым сложным и требует знаний определённых методов настройки [1]. Методы подбора данных коэффициентов можно разделить на ручные и автоматические [19]. В данной работе рассмотрим самый распространённый ручной подхода к настройке (метод Циглера-Никольса) и встроенный метод блока PID Controller.

В построенных системах управления, ПИД-регулятор выполняет функцию выравнивания робота вдоль линии, а П-регулятор выполняет функцию ускорения робота, когда он находится на прямой.

Таким образом, эффективностью подобранных параметров ПИД-регулятора, можно считать период T_θ и амплитуду A_θ угла поворота θ(t) робота при движении по прямой линии. Для визуальной оценки данных параметров воспользуемся блоком Scope, подключённому к Transform Sensor отслеживающему угол поворота робота относительно осей мира.

Для подбора коэффициентов воспользуемся методом Циглера-Никольса. Данный метод использует следующий алгоритм для подбора коэффициентов:

1. Устанавливаем нулевые значения коэффициентов.

2. Выбираем идеальный вариант работы системы (в нашем случае желаемый угол θ = 0).

3. Начинаем увеличивать пропорциональный коэффициент на некоторое значение и отслеживаем поведение системы.

4. При значении $K_P = \widetilde{K}$ возникают незатухающие колебания.

5. Записываем значение K_P и период колебаний системы $\widetilde{T_\theta}$.

6. Рассчитываем коэффициенты, по следующим формулам:

$$\begin{cases} K_P = 0{,}6 \cdot \widetilde{K}, \\ K_I = \dfrac{2 \cdot K_P}{\widetilde{T_\theta}}, \\ K_D = \dfrac{K_P \cdot \widetilde{T_\theta}}{8}. \end{cases} \tag{3.1}$$

При $K_P = 1$ получаем следующий график зависимости угла поворота робота от времени, представленный на рисунке 3.16. Данное значение удовлетворяет требованию возникновения незатухающих колебаний, период данных колебаний составляет примерно 0,0035с. Проделаем тоже самое для значений $K_P = 2 - 4$. При данных значениях так же возникают незатухающие колебания, занесём полученные данные в таблицу 2.

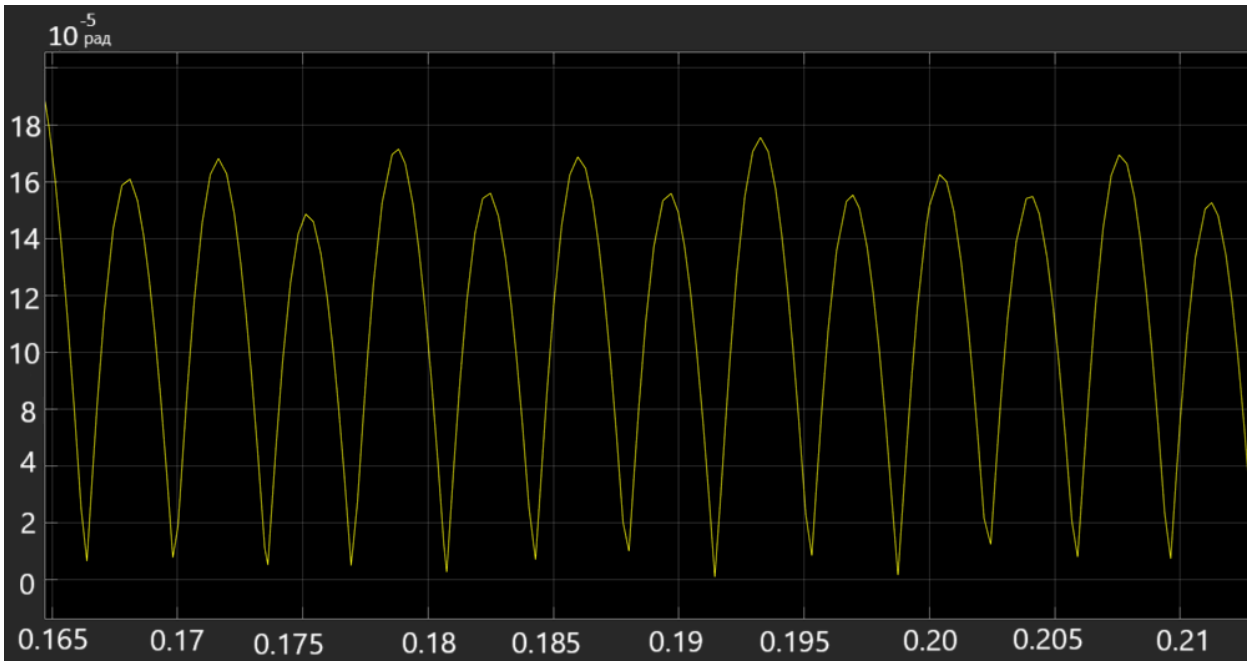

Рисунок 3.16 – График зависимости угла поворота робота (ось ординат) от времени (ось абсцисс) при использовании коэффициентов $K_P = 1, K_I = K_D = 0$

Таблица 2 – Полученные значения периодов устойчивых колебаний при различных пропорциональных коэффициентах

$\widetilde{K}$	T_θ, c
1	0,007
2	0,016
3	0,022
4	0,032

Проведём расчёты по формулам (3.1) и получаем следующие конфигурации ПИД-регулятора, представленные в таблице 2.

Таблица 3 – коэффициенты ПИД-регулятора, полученные методом Циглера-Никольса

№	K_P	K_I	K_D
1	0,6	171,4286	0,000526
2	1,2	150	0,0024
3	1,8	163,6364	0,00495
4	2,4	150	0,0096

Подставим данные конфигурации в ПИД-регулятор и проведём испытания.

В результате испытаний все четыре конфигурации позволили роботу пройти трассу, лучше всего показала себя четвёртая конфигурация, так как при ней робот отклоняется на наименьшую амплитуду. Полученный график зависимости угла поворота робота от времени, для четвёртой конфигурации представлен на рисунке 3.17.

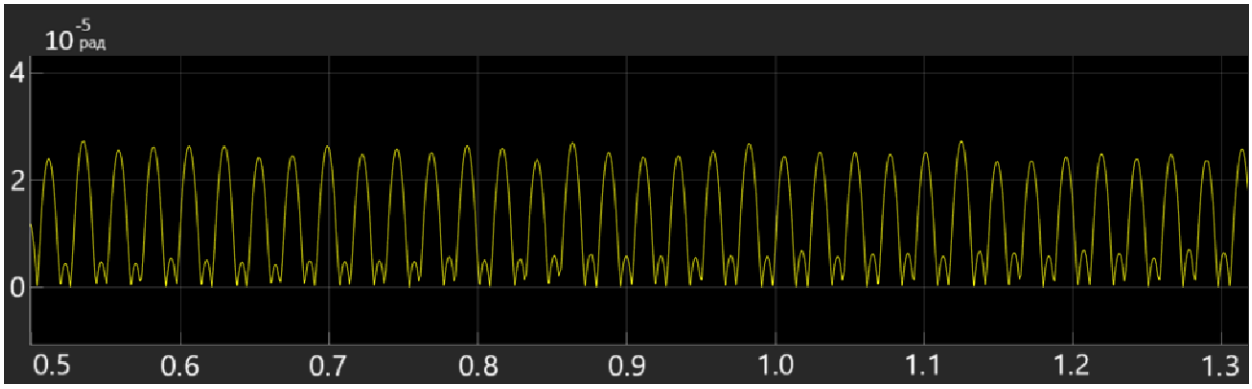

Рисунок 3.17 – График зависимости угла поворота робота (ось ординат) от времени (ось абсцисс) при использовании коэффициентов ПИД-регулятора, полученных методом Циглера-Никольса

Можно продолжать подбор лучших коэффициентов вручную, постепенно меняя каждый коэффициент ПИД-регулятора, однако блок PID Controller предлагает встроенную автонастройку, основанную на передаточной функции [10]. Применяя её получаем следующие коэффициенты: $K_P = 2.06633934981803$, $K_I = 170.342133712532$ и $K_D = 0.00475937486923523$. При данной конфигурации, робот проходит трассу с графиком угла поворота от времени, представленным на рисунке 3.18.

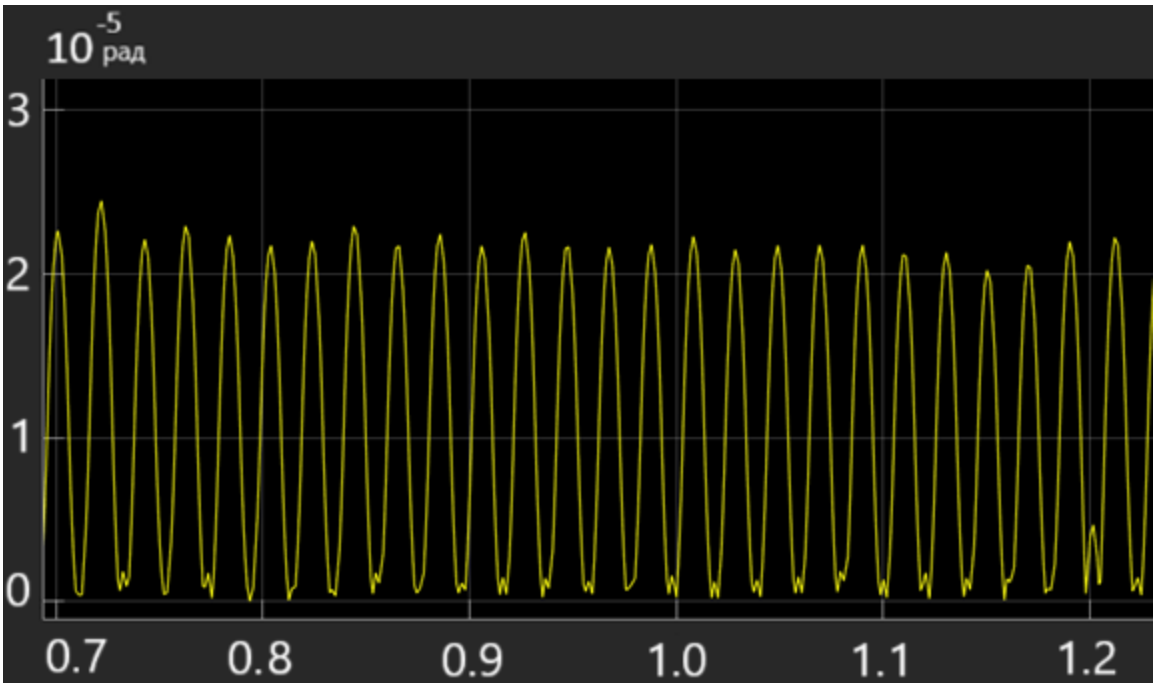

Рисунок 3.18 – График зависимости угла поворота робота (ось ординат) от времени (ось абсцисс) при использовании коэффициентов ПИД-регулятора, полученных автонастройкой

Далее будем использовать полученные автонастройкой коэффициенты для настройки П-регулятора. Так как П-регулятор выполняет функцию ускорения, основным критерием для его настройки является следующее неравенство:

$$\omega_{const} + u_{1_{max}} + u_{2_{max}} \leq \omega_{max}$$

где ω_{max} – максимальная угловая скорость робота;

ω_{const} – начальная скорость;

$u_{1_{max}}, u_{2_{max}}$- максимальное управляющее воздействие ПИД-регулятора.

Произведём замеры $u_{1_{max}}$ при выбранных коэффициентах. Получаем $u_{1_{max}} = 7{,}69\ \frac{\text{рад}}{\text{с}}$.

Подставим значения и преобразуем неравенство:

$$u_{2_{max}} \leq 12{,}2539 \frac{\text{рад}}{\text{с}}.$$

Исходя из формулы (2.13) и параметров модели, значение ошибки для П-регулятора, лежит в интервале [0,1]. Так как П-регулятор вырабатывает управляющее воздействие пропорционально, с коэффициентом K_{P_2}, для того чтобы робот двигался по прямой с максимальной возможной, с учётом ПИД-регулятора, скоростью, значение коэффициента K_{P_2} должно быть равно 12,2539.

Полученные выше коэффициенты для ПИД-регулятора могут быть использованы в обеих системах.

3.5 Реализация нейроэволюционного алгоритма

Для программной реализации нейроэволюционного алгоритма было решено использовать язык программирования Python, вместе с библиотеками NEAT-Python и Pygame.

Данный выбор был обусловлен удобным функционалом этой библиотеки, так как при её использовании создаётся конфигурационный файл, в котором можно удобно задавать все необходимые параметры нейронных сетей и генетического алгоритма. А, так же данная библиотека предоставляет функционал для удобного сохранения текущей популяции и геномов.

Для начала настроим конфигурационный файл NEAT-Python. Рассмотрим основные параметры, которые в нём можно задать:

1. fitness_criterion = max. Данным параметром задаётся то, как на основе значений fitness_i будут выявляться лучшие особи поколения. В нашем случае особи поколения, это особи с наибольшим fitness_i.
2. pop_size = 30. Количество машин в поколении.
3. no_fitness_termination = True. Данным параметром можно задать остановку обучения при достижении определённого значения fitness_i. В нашей задаче это не требуется, так как критерий остановки мы зададим сами, как и говорилось ранее.
4. activation_default = tanh. Функция активации нейронов.
5. aggregation_default = mean. Функция обработки входных значений (среднее арифметическое).
6. bias_init_mean/response_init_mean/weight_init_mean = 0.

Коэффициент сдвига μ = 0, для нормального распределения (для начальной инициализации bias/response/weight).

7. bias_init_stdev/response_init_stdev/weight_init_stdev = 1. $\sigma^2 = 1$, также для нормального распределения.

8. feed_forward = True. Указывает на то, что нейросети прямого распространения ошибки.

9. initial_connection = full_direct. Указывает на наличие всех связей между нейронами слоёв.

10. bias/response/weight_max/min_value = 30/-30. Задаёт параметр максимального/минимального значения, которые могут преобрести параметры нейронных сетей.

11. elitism = 5. Число лучших особей, которые будут выбраны для производства нового поколения.

Стоит отметить, что конфигурационный файл NEAT-Python имеет ещё много значений которые можно задать, но в нашем случае они были выставлены таким образом, чтобы подходить под постановку задачи и не играют важной роли, поэтому их рассматривать не будем.

Создадим класс Robot с необходимыми для работы методами, вот некоторые из них:

1. __init__. Конструктор класса, вызываемый при создании объекта Robot, в нём задаются начальные параметры машины. Основные из них:
   - self.speed = 5 – скорость машины ϑ_0.
   - self.is_alive = True – жива ли машина.
   - self.distance = 0 – пройденное расстояние.
   - self.pos = [100, 500] – начальные координаты x_0, y_0.
   - self.angle = 0 – начальный угол поворота φ_0.
   - self.time = 0 – время движения (количество дискретов времени в которое, машина совершала действия).
   - self.is_finished = False – достигла ли машина финиша.
2. is_live. Проверка жива ли машина. Если машина на белом цвете – жива, на зелёном – умерла, красный – достигла финиша.
3. compute_radars. Вычисляет значения радаров r_i, а также координаты точек, для их отрисовки.
4. draw. Отображает машину на трассе.
5. get_data. Возвращает показания с радаров в виде массива $[r_0, r_1, r_2, r_3, r_4, \vartheta]$.
6. get_reward. Возвращает значение fitness для робота, вычисляемое по формуле (2.17).

Основной цикл программы разместим в классе Traning. Он имеет два метода:

1. __init__. Его основной функцией является создание экземпляра класса neat.Population, с необходимой конфигурацией и вызов функции run:

config = neat.config.Config(neat.DefaultGenome, neat.DefaultReproduction, neat.DefaultSpeciesSet, neat.DefaultStagnation, config_path)

p = neat.Population(config)

p.run(self.run_generation)

2. run_generation. Основной метод, в котором и происходит обучение.

Рассмотрим подробнее основные элементы процесса работы run_generation.

Создание экземпляров нейросетей и машин:

```
nets = []
robots = []
for _, g in genomes:
    net = neat.nn.FeedForwardNetwork.create(g,
    config)
    nets.append(net)
    g.fitness = 0
    robots.append(Robot())
```

Создаём окно Pygame и отображаем трассу:

```
image = pygame.image.load(image_path)
image = pygame.transform.scale(image, (IMAGE_WIDTH,
IMAGE_HEIGHT))
pygame.init()
screen = pygame.display.set_mode((IMAGE_WIDTH,
IMAGE_HEIGHT + 50))
pygame.display.set_caption("RobotAI")
clock = pygame.time.Clock()
```

Далее процесс выбора действий:

```
for i, robot in enumerate(robots):
    robot.compute_radars(image)
    robot.is_live(image)
    if robot.is_alive:
      output = nets[i].activate(robot.get_data())
      robot.angle += math.floor(output[0]*10)
      if robot.speed + math.floor(output[1]*5) >= 5:
          robot.speed += math.floor(output[1]*5)
```

И процесс совершения действия, включает изменение координат машины по формулам (2.15):

```
for i, robot in enumerate(robots):
   robot.compute_radars(image)
   robot.is_live(image)
   if robot.is_alive:
```

```
            robots_left += 1
            robot.pos[0] -= math.sin(math.radians(360 -
                            robot.angle)) * robot.speed
            robot.pos[1] -= math.cos(math.radians(360 -
                            robot.angle)) * robot.speed
            robot.distance += robot.speed
            robot.time += 1
```

Так же необходимо подсчитать награды и передать их генетическому алгоритму:

```
for i, robot in enumerate(robots):
    genomes[i][1].fitness += robot.get_reward()
```

По окончанию если не был достигнут критерий остановки, то функция run запускается снова, предварительно создав новое поколение алгоритмом neat.

3.6 Обучение нейронной сети

Обучение будем проводить последовательно на трёх трассах, в порядке увеличения их сложности, для получения генома способного пройти трассы различной сложности. С учётом возможности робота ускоряться, будем так же смотреть на скорость прохождения трассы. Трассы, используемые для обучения представлены на рисунках 3.19, 3.20, 3.21.

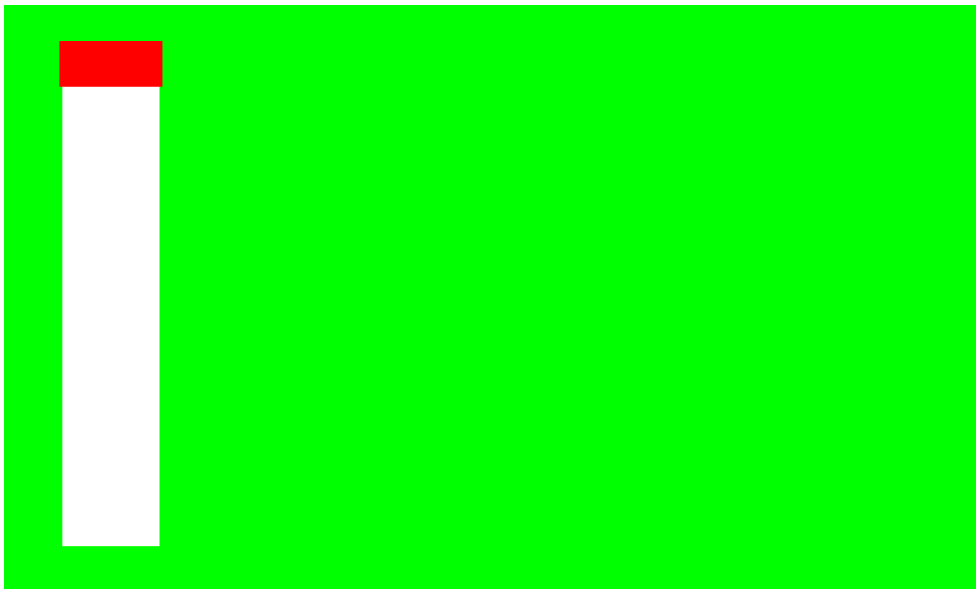

Рисунок 3.19 – Первая трасса

Рисунок 3.20 – Вторая трасса

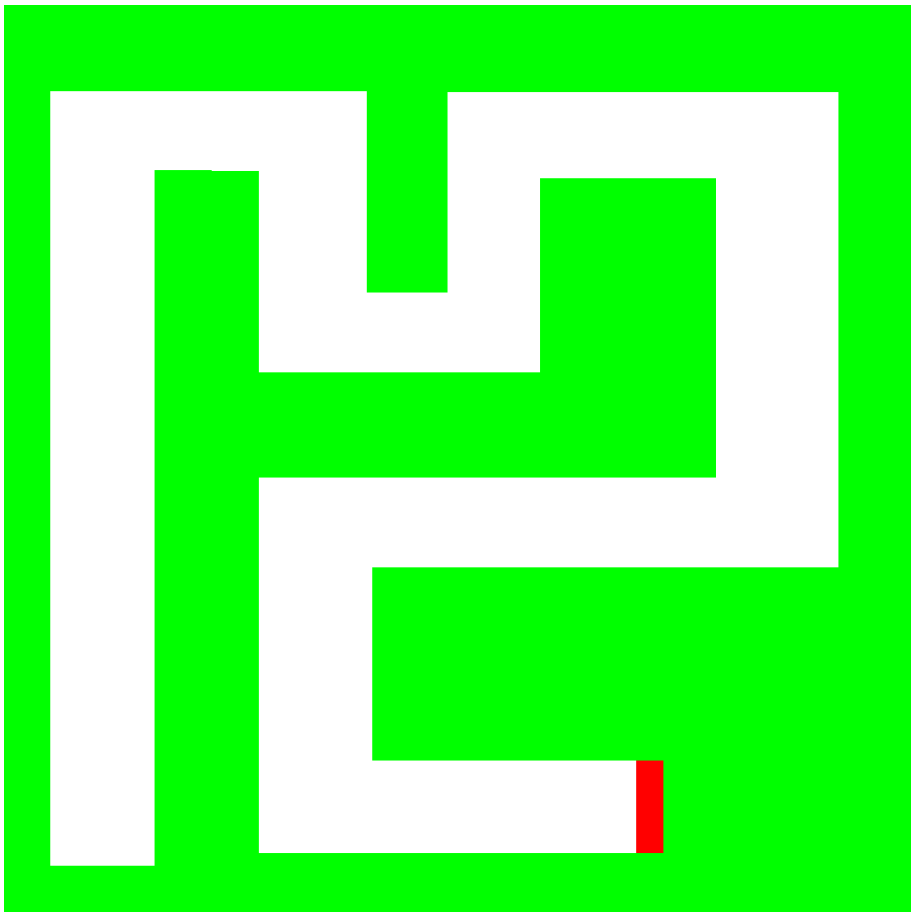

Рисунок 3.21 – Третья трасса

При обучении роботов на данных трассах, производилась запись времени прохождения самого быстрого робота в поколении, начиная с момента достижения роботами финиша. На основании полученных данных получаем графики времени прохождения трасс, представленные на рисунках 3.22, 3.23, 3.24.

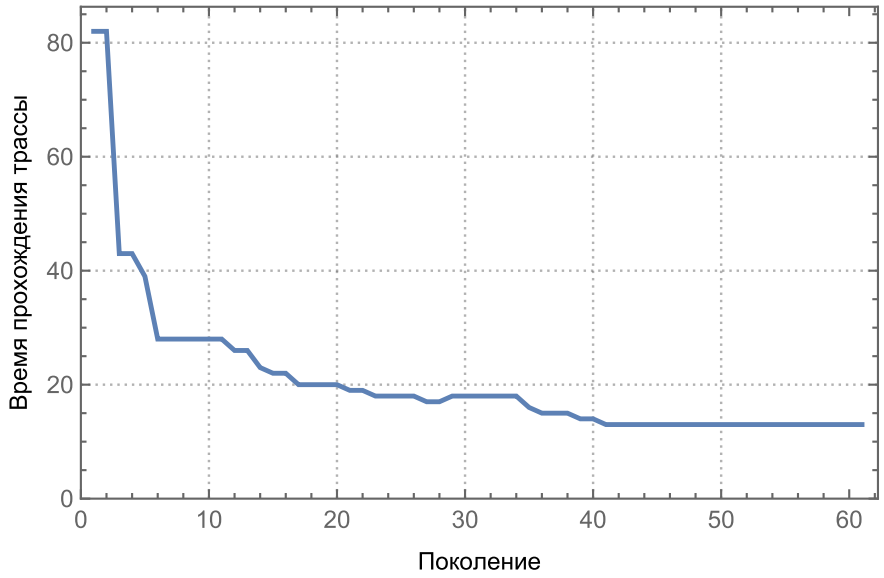

Рисунок 3.22 – График времени прохождения первой трассы

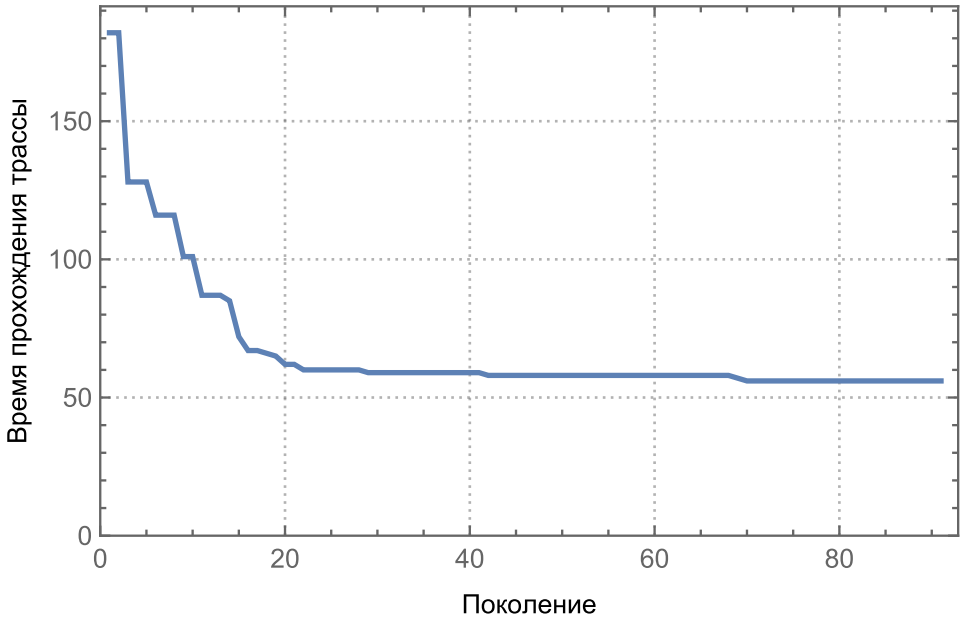

Рисунок 3.23 – График времени прохождения второй трассы

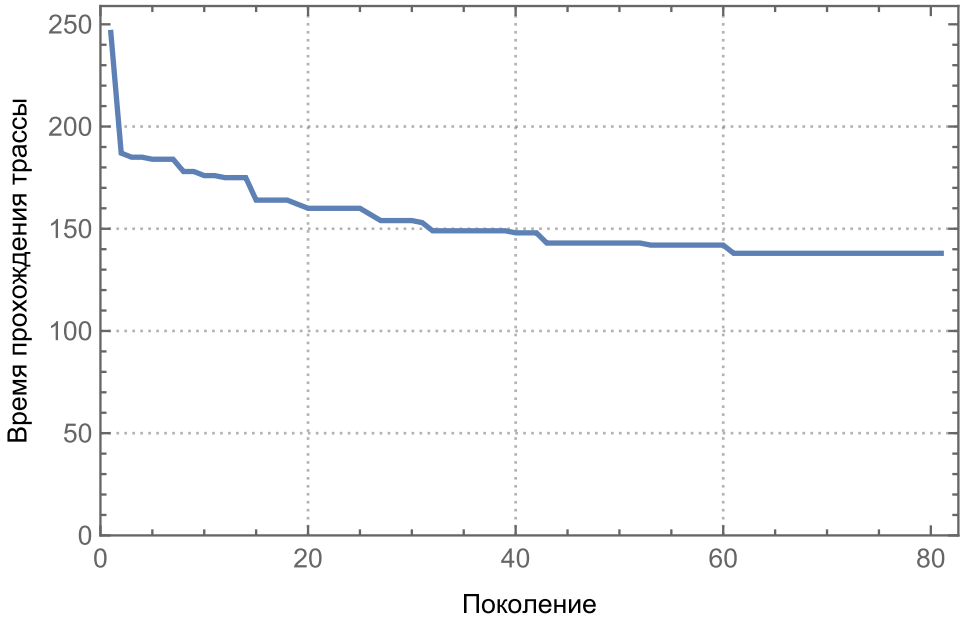

Рисунок 3.24 – График времени прохождения третьей трассы

Минимумы графиков 3.22, 3.23, 3.24 соответствуют наилучшему результату среди поколений:

- Для первой трассы это 13 секунды, по сравнению с 82 секундами начальными;
- Для второй трассы это 56 секунд, по сравнению с 182 секундами начальными;
- Для третьей трассы это 138 секунд, по сравнению с 246 секундами начальными.

3.7 Реализация нейроэволюционной системы управления

MATLAB Simulink поддерживает возможность запуска функций языка Python и подключения его библиотек. Таким образом, лучший геном (нейронная сеть с лучшими параметрами), полученный выше, может быть сериализован в программе для обучения и десериализован в проекте MATLAB Simulink.

Для начала сериализуем лучший геном при помощи библиотеки pickle:

```
with open("best_genome.pkl", "wb") as f:
    pickle.dump(best_genome, f)
```

Получаем файл best_genome.pkl с лучшим геномом.

Используя данный файл, напишем функцию activate. Данная функция принимает входные значения нейронной сети, дисериализует геном из файла и возвращает выходные значения. Код данной функции имеет следующий вид:

```
def activate(data):
    file = "best_genome.pkl"
    with open(file, "rb") as f:
        best_genome = pickle.load(f)
    config_path = "config-feedforward.txt"
    config = neat.Config(neat.DefaultGenome,
             neat.DefaultReproduction,
             neat.DefaultSpeciesSet,
             neat.DefaultStagnation, config_path)
     net = neat.nn.FeedForwardNet.create(best_genome,
           config)
    return net.activate(data)
```

Полученную функцию в дальнейшем будем вызывать в MATLAB Simulink.

В связи с большим объёмом вычислений при данном подходе, визуализация на 3d полигоне становится очень медленной. Для облегчения вычислений и отображения, воспользуемся Mobile Robotics Simulation Toolbox. Данный набор инструментов предоставляет готовые решения для работы с мобильными роботами в среде MATLAB Simulink. Из этого набора нам понадобятся следующие блоки:

- Differential Drive Simulation. Данный блок реализует вычисления по формулам N, принимая на вход угловые скорости колёс, и возвращая текущие координаты робота $[x, y, \theta]$ (реализация формул (2.4), с начальными условиями (2.5)). Для его работы в настройках блока необходимо задать расстояние между колёсами, радиус колёс и начальное положение робота.
- Robot Visualizer. Данный блок в реальном времени отображает положение и траекторию движения робота на карте, принимая на вход его координаты. Для его работы необходимо задать радиус основания робота и массив, соответствующий изображению карты.
- Lidar Sensor. Данный блок представляет собой аналоговый лидар, принимающий координаты робота и возвращающий показания лидара. Для его работы необходимо задать углы, в которых будет рассчитываться расстояние до стен и массив, соответствующий изображению карты.

Одним из входных параметров описанной выше сети является линейная скорость робота в пикселях, реализуем подсистему для её вычисления по формуле N. Вид данной подсистемы представлен на рисунке 3.25.

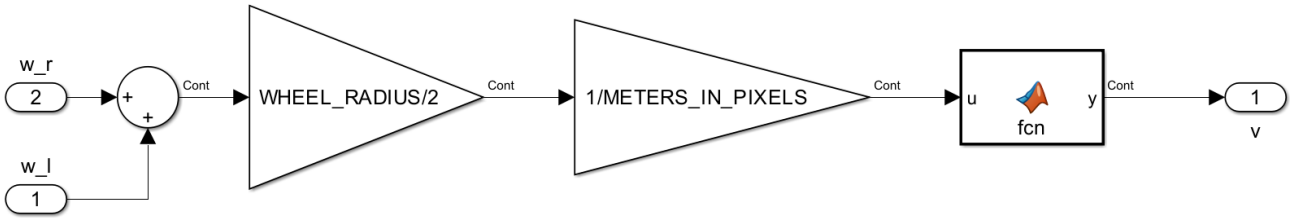

Рисунок 3.25 – Подсистема для вычисления линейной скорости

Реализуем вызов функции activate в блоке MATLAB Function. Вид данной функции представлен на рисунке 3.26.

```
function output = net(data)
    coder.extrinsic('py.importlib.import_module');
    coder.extrinsic('py.get_params.activate');
    coder.extrinsic('cell');
    coder.extrinsic('cell2mat');
    coder.extrinsic('double');
    output = zeros(1, 2);
    py.importlib.import_module('neat');
    py.importlib.import_module('get_params');
    y_python = py.get_params.activate(data);
    disp(y_python)
    y_converted = coder.nullcopy(output);

    if ~isempty(y_python)
        y_matlab = double(cell2mat(cell(y_python)));
        y_converted(:) = y_matlab(1:numel(output));
    end
    output = y_converted;
end
```

Рисунок 3.26 – Функция, реализующая работу нейронной сети

Так же реализуем функцию Convert Data приводящую показания лидара к необходимому для входа нейронной сети виду.

При обучении нейронной сети, использовалась линейная скорость робота, поэтому необходимо реализовать функцию Update Angular Velocities, реализующую действие робота и получение угловых скоростей колёс по следующим формулам обратной кинематики:

$$\begin{cases} \omega_R = \dfrac{2\vartheta + \omega \cdot L}{2r}, \\ \omega_L = \dfrac{2\vartheta}{r} - \omega \cdot r, \end{cases} \tag{3.2}$$

где ω_R и ω_L – угловые скорости колёс;

ϑ и ω – линейная и угловая скорость робота;

L – расстояние между колёсами;

r – радиус колёс.

Вид функции Update Angular Velocities, реализующей формулы (3.2), представлен на рисунке 3.27.

```
function [wl,wr] = fcn(output,last_v,L,r)
w = output(1)*(-15)*pi/180;
delta_v = output(2)*5;
if last_v+delta_v <= 5
    v = 5;
elseif last_v+delta_v > 105
    v = 105;
else
    v = last_v+delta_v;
end
v = v*0.005;
wr = (2*v + w*L)/(2*r);
wl = 2*v/r - wr;
end
```

Рисунок 3.27 – Функция Update Angular Velocities

Соединяем полученные элементы и получаем нейроэволюционную систему управления, вид которой представлен на рисунке 3.28.

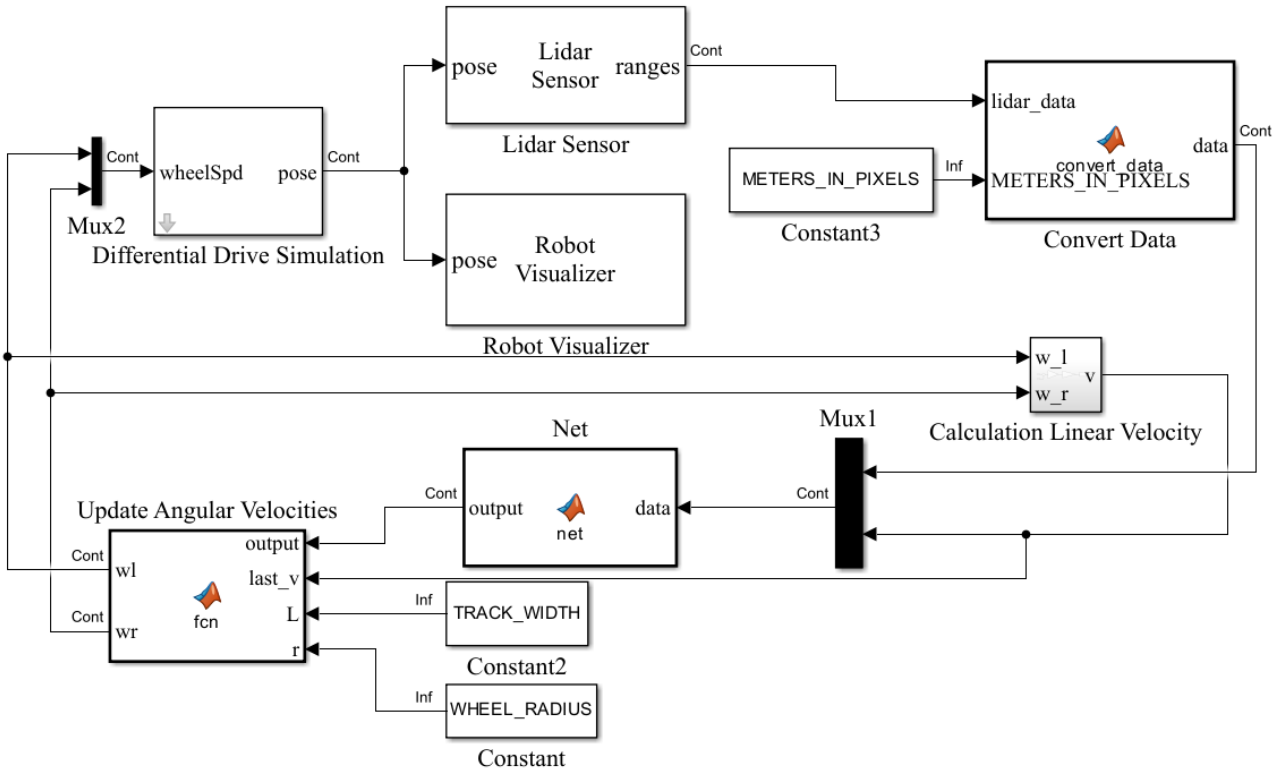

Рисунок 3.28 – Нейроэволюционная система управления

При запуске данной системы, в окне Robot Visualizer, наблюдаем, что робот проезжает трассу по траектории, представленной на рисунке 3.29.

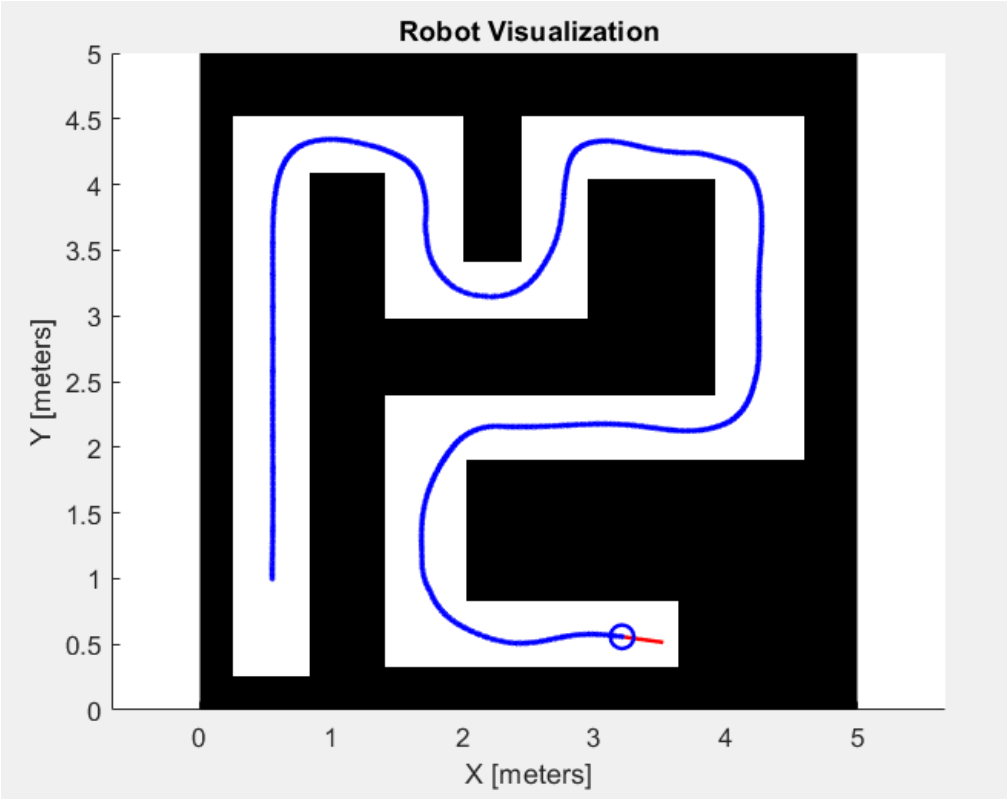

Рисунок 3.29 – Вид окна Robot Visualizer

Так же стоит отметить, что робот проходит трассу с той же скоростью, что была описана при обучении.

3.8 Выводы по третьей главе

В рамках данной главы была проведена разработка цифрового двойника мобильного робота RoboCake с использованием инструментальной среды MATLAB Simulink. Цифровой двойник, соответствующий функциональным характеристикам физического прототипа робота, что обеспечивает возможность проведения испытаний и отладки алгоритмов управления в виртуальной среде. Для обеспечения полноты и достоверности испытаний был создан испытательный полигон.

В процессе разработки были исследованы и протестированы различные системы управления цифрового двойника:

1. Ручную систему управления, которая предоставляет оператору возможность интерактивного задания скорости вращения каждого колеса в режиме реального времени через пульт управления. Данная система была использована для проведения начальных тестов и отладки поведения модели.
2. Две системы управления, основанные на применении пропорционально-интегрально-дифференциальных (ПИД) регуляторов. Первая система обеспечила стабильное движение робота с минимальными отклонениями от заданной траектории, в то время как вторая система, дополненная пропорциональным регулятором скорости, позволила достигать более высоких скоростей на прямолинейных участках трассы, при этом сохраняя устойчивость движения.
3. Нейроэволюционную систему управления, которая использует данные с лидаров и предварительно обученную нейронную сеть. Эта система продемонстрировала способность к адаптивному поведению и эффективному прохождению сложных маршрутов. Для её реализации было разработано специализированное программное приложение, реализующее алгоритмы нейроэволюции. В результате была получена обученная модель, способная принимать оптимальные решения в условиях неопределённости.

Анализ результатов настройки ПИД-регуляторов, проведённой с использованием трёх различных методов, показал, что наилучшие результаты были достигнуты при применении встроенного алгоритма MATLAB, который использует передаточную функцию объекта управления. Данный подход позволил определить оптимальные параметры регуляторов, что обеспечило устойчивое поведение робота при следовании по линии.

На основании проведённых исследований можно сделать следующие ключевые выводы:

- Цифровой двойник является эффективным инструментом для тестирования и разработки систем управления мобильными роботами.
- Применение различных подходов к управлению, от классических ПИД-регуляторов до нейросетевых моделей, способствует повышению адаптивности и производительности робототехнических систем.
- Инструменты MATLAB Simulink предоставляют широкий спектр возможностей для моделирования, параметрической настройки и визуализации функциональных характеристик систем управления.

Результаты, полученные в рамках данной главы, подтверждают целесообразность и эффективность применения цифрового двойника в качестве платформы для проектирования, тестирования и оптимизации систем управления мобильными роботами.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы был разработан цифровой двойник мобильного робота и системы управления его движением, что позволяет сделать выводы о практической эффективности использования цифровых двойников в прикладных задачах мобильной робототехники.

Построенная математическая модель движения, основанная на физической модели мобильного робота RoboCake, позволяет рассчитывать его координаты и угол поворота. Полученную модель можно адаптировать для мобильных роботов, со схожими характеристиками и использовать для построения цифровых двойников.

Разработанный цифровой двойник, в среде MATLAB Simulink, удовлетворяет поставленным требованиям и подходит для создания систем управления мобильным роботом RoboCake. Полученные знания подтверждают эффективность использования цифровых двойников для построения систем управления мобильным роботом.

Полученные системы управления можно использовать для управления движением мобильных роботов, так как MATLAB Simulink позволяет переносить эти системы на платы серии Arduino.

Анализ эффективности применения ПИД-регуляторов для задач движения мобильного робота, демонстрирует их основные преимущества, такие как: гибкая настройка скорости движения и поддержание устойчивости системы.

Анализ эффективности применения нейроэволюции для построения систем управления демонстрирует эффективность подхода и его основные преимущества: адаптивность к различным трассам и способность находить оптимальные маршруты движения.

Результаты работы предназначены для практического использования в различных отраслях, применяющих мобильную робототехнику в своей работе. Они могут использоваться с целью построения и настройки систем управления мобильными роботами, анализа их поведения и пригодность для различных задач. Применения технологии создания цифрового двойника, позволяет значительно сократить время и материальные средства, при работе с мобильными роботами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Автоматизированная настройка ПИД-регулятора для объекта управления следящей системы с использованием программного пакета MATLAB Simulink / А. В. Филиппов [и др.] // Наука, техника и образование. – 2015. – № 12(18). – С. 53–59.

2. Мюллер А. Введение в машинное обучение с помощью Python. Руководство для специалистов по работе с данными. / А. Мюллер, С. Гвидо - Москва, 2017-393с.

3. Гладков Л. А., Курейчик В. В., Курейчик В.М. Генетические алгоритмы / Под ред. В.М. Курейчика. — 2-е изд., исправл. и доп. — М.: ФИЗМАТЛИТ, 2010. — 368 с.

4. Генетические алгоритмы на Python / Э. Вирсански : пер. с англ. / А. А. Слинкина; — М.: ДМК Пресс, 2020. — 286с.

5. Документация на серию QRE1113 [Electronic resource]. – Mode of access: https://www.compel.ru/itempdf/d0c1a5f6f0546f38/ps/fair~qre1113.pdf. – Date access: 15.05.2025.

6. Компьютерные модели и моделирование. Цифровые двойники изделий. Общие положения: ГОСТ Р 57700.37- 2021; введ. РФ 01.01.22. — Москва: Федеральное агентство по техническому регулированию и метрологии.

7. Арнольд В. И. Математические методы классической механики / В. И. Арнольд - М.: Главная редакция физико-математической литературы изд-ва «Наука», 1974 - 432с.

8. Ким, Т. Ю. Оптимизация коэффициентов ПИД-регулятора системы управления дви жением мобильного робота по цветоконтрастной линии на основе генетического алгоритма / Т. Ю. Ким, Г. А. Прокопович // Информатика. – 2021. − Т. 18, № 4. – С. 53–68.

9. Сейдж, Э. П. Оптимальное управление системами / Э. П. Сейдж, Ч. С. Уайт ; пер. с англ. – М. : Радио и связь, 1982. – 392 с.

10. Варламов, И. ПИД-EXPERT – автоматизация автоматизации / И. Варламов, П. Зубов // Control Engineering. – 2017. – № 2(68). – С. 98–101.

11. Теория автоматического управления: учебно-методическое пособие для студентов специальностей 1-53 01 04 «Автоматизация и управление теплоэнергетическими процессами», 1-43 01 04 «Тепловые электрические станции», 1-43 01 08 «Паротурбинные установки атомных электрических станций», 1-53 01 01 «Автоматизация технологических процессов и производств» / Г. Т. Кулаков [и др.]; под общ. ред. Г. Т. Кулакова. – Минск: БНТУ, 2017. – 133 с.

12. Умное производство. Модель эталонной архитектуры индустрии 4.0 (RAMI 4.0): ГОСТ Р 59799-202; введ. РФ 30.04.2022. — Москва: Федеральное агентство по техническому регулированию и метрологии.

13. Рудинский, И.Д. Элементарное введение в технологию нейронных сетей с примерами программ / И. Д. Рудинский. - М.: Горячая линия - Телеком, 2011. - 408с.

14. Ansys Twin Builder [Electronic resource]. – Mode of access: https://www.ansys.com/products/digital-twin/ansys-twin-builder. – Date access: 17.05.2025.

15. Arduino – Hardware Support – MATLAB & Simulink [Electronic resource]: The MathWorks. – Mode of access: https://www.mathworks.com/discovery/industry-4-0.html. – Date access: 02.05.2025.

16. Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models., New York, June 2020/ Institute of Electrical and Electronics Engineers; ed.: R. Minerva, G. M. Lee, N. Crespi.

17. Micheal Lanham. Evolutionary Deep Learning. Genetic algorithms and neural networks / Manning Shelter Island — United States of America — 2023.

18. From Simulation to Experimentable Digital Twins Simulation-based Development and Operation of Complex Technical Systems., Germany, Nov. 2016/ Institute for Man-Machine Interaction Engineers; ed.: Michael Schluse, Juergen Rossmann.

19. O'Dwyer, A. Handbook of PI and PID Controller Tuning Rules / A. O'Dwyer. – London : Imperial College Press, 2003. – 564 p.

20. Industry 4.0 [Electronic resource]: The MathWorks. – Mode of access: https://www.mathworks.com/discovery/industry-4-0.html. – Date access: 06.05.2025.

21. NEAT-Python's documentation [Электронный ресурс]. — Режим доступа: https://neat-python.readthedocs.io/en/latest/. — Дата доступа: 02.05.2025.

22. Open Modelica [Electronic resource]. – Mode of access: https://openmodelica.org/. – Date access: 08.05.2025.

23. Simulink [Electronic resource]: The MathWorks. – Mode of access: https://www.mathworks.com/products/simulink.html. – Date access: 10.05.2025.

24. World Robotics 2024 Report [Electronic resource]: International Federation of Robotics. – Mode of access: https://ifr.org/downloads/press2018/2024-SEP-24_IFR_World_Robotics_2024_-_global_market.pdf. – Date access: 01.05.2025.