

**МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)**

Институт радиоэлектроники и информационных технологий
Направление подготовки (специальность) 09.03.01 Информатика и вычислительная техника
Направленность (профиль) образовательной программы Вычислительные машины, комплексы, системы, сети
Кафедра Вычислительные системы и технологии


ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

бакалавра

Студента Шунина Кирилла Максимовича группы 16-В-2


на тему: «Программная система управления движением мобильного робота.»

СТУДЕНТ:


(подпись) Шунин К.М.
(фамилия, и., о.)

«2» июля 2020г.
(дата)

РУКОВОДИТЕЛЬ:


(подпись) Гай В.Е.
(фамилия, и., о.)


«2» июля 2020г.
(дата)

РЕЦЕНЗЕНТ:

(подпись) (фамилия, и., о.)

(дата)

ЗАВЕДУЮЩИЙ КАФЕДРОЙ:


(подпись) Жевнерчук Д.В.
(фамилия, и., о.)

«2» июля 2020г.
(дата)

КОНСУЛЬТАНТЫ:

1. По _____

(подпись) (фамилия, и., о.)

(дата)

2. По _____

(подпись) (фамилия, и., о.)

(дата)

3. По _____

(подпись) (фамилия, и., о.)

(дата)

ВКР защищена _____
(дата)

протокол № _____
с оценкой _____

**МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)**

Кафедра «Вычислительные системы и технологии»

УТВЕРЖДАЮ

Зав. кафедрой



Жевнерчук Д.В.

«12» мая 2020 г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

по направлению подготовки (специальности) 09.03.01 Информатика и вычислительная техника студенту Шунину К.М. группы 16-В-2

1. Тема «Программная система управления движением мобильного робота»

(утверждена приказом по вузу от № 867/5 от 15.04.2020)

2. Срок сдачи студентом законченной работы 2 июля 2020г

3. Исходные данные к работе: описание робота ELCBot

4. Содержание расчетно-пояснительной записки

Введение

1. Требования к продукту

2. Анализ технического задания

3. Разработка цифрового двойника

4. Разработка системы управления движением

5. Тестирование системы

Заключение

Перечень сокращений

Список литературы

5. Перечень графического материала (с точным указанием обязательных чертежей)

Презентация (Предпосылки, Цели и задачи, Окружение цифрового двойника, Цифровой двойник, Процесс моделирования, Система управления движением, Патрулирование по периметру, Движение вдоль стены, Подбор коэффициентов PID регулятора, Видеодемонстрация)


6. Консультанты по ВКР (с указанием относящихся к ним разделов)

Нормоконтроль Жевнерчук Д.В.

7. Дата выдачи задания «13» апреля 2020 г.

Код и содержание Компетенции	Задание	Проектируемый результат	Отметка о выполнении
ПК-1 Способность разрабатывать модели компонентов информационных систем, включая модели баз данных и модели интерфейсов "человек - электронно-вычислительная машина"	Разработка системы взаимодействия человека с цифровым двойником посредством управления при помощи клавиатуры	Возможность ручного управления движением цифрового двойника	Выполнено
ПК-2 способностью разрабатывать компоненты аппаратно-программных комплексов и баз данных, используя современные инструментальные средства и технологии программирования	Разработка цифрового двойника реального робота ELCBot, а также разработка алгоритма движения для патрулирования помещения	Цифровой двойник с высокой точностью, виртуальное окружение цифрового двойника, режим автоматического патрулирования виртуального окружения	Выполнено
ПК-3 способностью обосновывать принимаемые проектные решения, осуществлять постановку и выполнять эксперименты по проверке их корректности и эффективности	Реализация PID регулятора с подобранными коэффициентами. Тестирование готовой системы.	PID регулятор настроен под различные ситуации и вся система протестирована и совершает полный круг по периметру виртуального окружения	Выполнено

Руководитель


(подпись)

Гай В.Е.

Задание принял к исполнению «13» апреля 2020 г.

Студент


(подпись)

Шунин К.М.

**МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)**

АННОТАЦИЯ

к выпускной квалификационной работе

по направлению подготовки (специальности) 09.03.01 Информатика и вычислительная техника студента Шунина К.М. группы 16-В-2
по теме «Программная система управления движением мобильного робота.»

Выпускная квалификационная работа выполнена на 55 страницах, содержит 8 таблиц, 24 рисунка, библиографический список из 6 источников, 3 приложения (дополнительно 14 страниц).

Актуальность: Выбранная задача актуальна и предоставляет возможности для безопасного и более простого внедрения информационных технологий в массы.

Объект исследования: робот ELCBot

Предмет исследования: алгоритм движения мобильного робота

Цель исследования: разработка программной реализации системы движения мобильного робота

Задачи исследования: выбрать симулятор роботов, разработать цифровой двойник реального робота ELCBot, смоделировать виртуальное окружение, написать программный алгоритм для ручного и автоматического управления движением

Методы исследования: Цифровой двойник и симулятор роботов

Структура работы:

Во введении рассказывается о цели работы, ее актуальности

В 1 разделе рассмотрены требования к ЭВМ, симулятору роботов, цифровому двойнику, системе управления движением

Во 2 разделе анализируется техническое задание, происходит выбор симулятора и языка программирования

В 3 разделе моделируется цифровой двойник и его окружение

В 4 разделе разрабатывается алгоритм движения цифрового двойника внутри окружения

В 5 разделе производится тестирование системы

В заключении проанализирована проделанная работа и сделаны выводы.

Выводы:

1. Разработан цифровой двойник
2. Смоделировано виртуальное окружение
3. Реализована система управления движением

Рекомендации:

1. Дальнейшее развитие проекта
2. Модернизирование существующего алгоритма системы управления движением







/ Шунин К.М.

«2» июля 2020 г.

подпись студента /расшифровка подписи

Оглавление

Введение.....	5
1. Требования к продукту	8
1.1. Назначение разработки и область применения	8
1.2. Технические требования	9
1.2.1. Требования к ЭВМ.....	9
1.2.2. Требования к симулятору роботов.....	9
1.2.3. Требования к цифровому двойнику ELCBot	9
1.2.4. Требования к системе управления движением	10
2. Анализ технического задания	11
2.1. Анализ аппаратного обеспечения	11
2.2. Выбор симулятора	11
2.3. Выбор языка программирования.....	16
2.4. Выбор среды разработки.....	18
3. Разработка цифрового двойника	20
3.1. Анализ реального прототипа	20
3.2. Создание окружения цифрового двойника	25
3.3. Создание цифрового двойника.....	27
4. Разработка системы управления движением	39
4.1. Декомпозиция объектов и написание ручного управления	39
4.2. Режим патрулирования	43
4.3. PID регулятор	46
4.3.1. Алгоритм движения вдоль стены.....	46
4.3.2. Настройка коэффициентов PID регулятора	49
5. Тестирование системы.....	52
Заключение	53
Перечень сокращений.....	54
Список литературы	55

					ВКР-НГТУ-09.03.01-(16-В-2)-016-2020 (ПЗ)			
Изм.	Лист	№ докум.	Подпись	Дата	Программная система управления движением мобильного робота Пояснительная записка	Лит.	Лист	Листов
Разраб.		Шунин К.М.		02.07				
Провер.		Гай В.Е.		02.07			4	55
Н. контр.				02.07		НГТУ кафедра ВСТ		
Утверд.		Жевнерчук Д.В.		02.07				

Введение

Система управления движением мобильного робота – одна из важнейших частей, которую необходимо реализовать перед демонстрацией продукта для дальнейшего развития. Именно от нее зависит возможность взаимодействия робота с окружающим миром. Без нее, робот, который по проекту должен иметь возможность передвигаться попросту не сможет этого делать.

У команды разработчиков ELCBot, в состав которой входит автор, после победы на конкурсе IT Проект Года 2018 среди студенческих проектов, был поставлен вопрос о дальнейшем продвижении проекта. ECLBot – это робот телеприсутствия, который подразумевает под собой возможность передвижения. Поэтому разработка системы управления движения встала на первое место в плане работы над проектом.

Режим самоизоляции в связи с пандемией COVID-19 продиктовал свои условия процесса работы над проектом. Поэтому было решено немного скорректировать начальную постановку задачи от реализации системы движения на реальном роботе в сторону создания цифрового двойника и написания контроллера к нему в специализированном симуляторе робота.

В ходе выполнения выпускной квалификационной работы предстояло проанализировать доступные симуляторы, в выбранном симуляторе создать цифровой двойник реального робота, настроить функциональное оснащение цифрового двойника и написать первую версию тестового контроллера, который позволяет управлять цифровым двойником с клавиатуры компьютера, а так же переключает робота в режим свободного патрулирования помещения.

Актуальность разработки заключается в том, что, если при экспортировании написанной прошивки реальный робот сможет

					ВКР-НГТУ-09.03.01-(16-В-2)-016-2020(ПЗ)	Лист
						5
Изм	Лист	№ докум.	Подп.	Дата		

беспрепятственно перемещаться по помещению, добавив еще несколько модулей получится модель передвижного помощника для первокурсников. Расположив такого ассистента, например, в шестом корпусе НГТУ получаем процесс консультации студентов без участия человека, что в рамках пандемии выглядит очень выгодно.

Есть необходимость в объяснении пункта, который был надиктован пандемией – цифровой двойник. Данный термин появился относительно недавно, с преобладанием на заводах так называемого «цифрового производства», когда участие человека в производстве какого-либо продукта сводится к контролированию роботизированных систем. Так же появлению цифровых двойников поспособствовало массовое внедрение в нашу жизнь интернет вещей, а в области производства промышленного интернета вещей, когда в любое устройство встраивается множество датчиков и сенсором, которые позволяют этому устройству взаимодействовать с пользователем. Назначение этих устройств поражает воображение, от умных автомобилей до привычных нам чайников.

Использование цифровых двойников причисляют к так называемому «Четвертой промышленной революции» - массовому внедрению в производство кибернетических систем (результатирующая система, полученная путем внедрения в физическую сущность вычислительных мощностей, подобным описанием обладает и термин IoT).

Цифровой двойник позволяет инженерам более качественно поддерживать разрабатываемый продукт, а также вести пост производственный анализ качества и оценку состояния как системы в целом, так и каждой его части в отдельности. На равне с этим, цифровой двойник позволяет существенно упростить процесс тестирования продукта, так как становится реальным перенести дорогостоящие и трудоемкие испытания,

					ВКР-НГТУ-09.03.01-(16-В-2)-016-2020(ПЗ)	Лист
						6
Изм	Лист	№ докум.	Подп.	Дата		

которые проводятся с готовым прототипом продукта в реальном окружении в цифровой мир, и проводить эти испытания там, но с меньшими затратами и большим охватом сценариев тестирования, вплоть до каких-то нереальных и опасных, если бы они проводились в реальном мире.

					ВКР-НГТУ-09.03.01-(16-В-2)-016-2020(ПЗ)	Лист
						7
Изм	Лист	№ докум.	Подп.	Дата		

1. Требования к продукту

1.1. Назначение разработки и область применения

Сама система управления движением мобильного робота предназначена для предоставления различных сценариев движения робота. От ручного управления с различных устройств передачи команд, до полностью автоматизированного передвижения без участия пилота. Данная разработка предназначена исключительно для робота теле присутствия ELCBot, разрабатываемого доцентом кафедры ВСТ Гай Василием Евгеньевичем и группой заинтересованных студентов.

Цифровой двойник реального робота ELCBot предназначен для моделирования поведения робота в окружении, максимально приближенном к реальности. Он позволит упростить разработку отдельных частей и модулей для модификации реального прототипа, потому что отпадет необходимость прямого доступа к реальному роботу. Цифровой двойник позволит смоделировать ситуации, которые никогда не могут произойти в реальной жизни и пронаблюдать возможные исходы таких ситуаций. Протестировать систему с максимальным уровнем безопасности, потому что это будет только цифровая симуляция.

					ВКР-НГТУ-09.03.01-(16-В-2)-016-2020(ПЗ)	Лист
						8
Изм	Лист	№ докум.	Подп.	Дата		

1.2. Технические требования

С учетом того, что работа над проектом перешла от реализации на реальном роботе к реализации на цифровой модели, то технические требования сводятся к ряду требований

1.2.1. Требования к ЭВМ

1. ОС с GUI.

2. аппаратное обеспечение, которое удовлетворяет требованиям как ОС, так и эмулятору, в котором будет выполнена работа.

3. необходимое периферийное оборудование, минимально: клавиатура, мышь, дисплей.

1.2.2. Требования к симулятору роботов

1. возможность работать с физикой объектов.

2. возможность написания прошивки на различных популярных в 2020 году языков программирования.

3. возможность создания собственного прототипа робота, а не только использование существующих.

4. высокая активность команды разработчиков симулятора (поддержка пользователей и выпуск новых версий продукта)

1.2.3. Требования к цифровому двойнику ELCBot

1. размеры и внешний вид максимально приближенный к оригиналу

2. периферийное оборудование в виде датчиков работы с окружающим миром должно точно соответствовать оригиналу

					ВКР-НГТУ-09.03.01-(16-B-2)-016-2020(ПЗ)	Лист
						9
Изм	Лист	№ докум.	Подп.	Дата		

3. окружение цифрового двойника должно быть из области его дальнейшего применения.

1.2.4. Требования к системе управления движением

1. возможность управления движением в ручном режиме с клавиатуры.

2. возможность перевода цифрового двойника в режим свободного патрулирования комнаты.

3. возможность цифрового двойника прямолинейно двигаться по периметру помещения.

4. движение цифрового двойника должно максимально задействовать периферийное оборудование для взаимодействия с окружающим миром.

					ВКР-НГТУ-09.03.01-(16-В-2)-016-2020(ПЗ)	Лист
						10
Изм	Лист	№ докум.	Подп.	Дата		

2. Анализ технического задания

2.3. Анализ аппаратного обеспечения

Работа выполнена на ноутбуке модели Dell Latitude 5500 на базе процессора Intel Core i5-8365U, видеопроцессором Intel UHD Graphics, 24 Гб оперативной памяти, 256 Гб SSD и установленной ОС Windows 10 Pro 64-bit. Данные характеристики полностью соответствуют техническим требованиям пункта 1.2.1

2.2. Выбор симулятора

Выбор симулятора для разработки цифрового двойника, пожалуй, является самым главным выбором всей работы. Именно от симулятора зависит точность и простота всей последующей разработки.

Проанализировав рынок в данной сфере, было найдено большое количество симуляторов. Полный список можно увидеть на рисунке 1.

Software ↕	Developers ↕	Development status ↕	License ↕	3D rendering engine ↕	Physics engine ↕	3D modeller ↕	Platforms supported ↕
Actin	Energid Technologies	Active	Proprietary	OpenGL	Proprietary	Proprietary	Windows, macOS, Linux, RTLinux, VxWorks, RTOS-32, and RTX. (QNX Planned)
ARS	RAL	Inactive	BSD	VTK	ODE	None	Linux, macOS, Windows
AUTOMAPPS	Convergent Information Technologies GmbH	Active	Proprietary	OpenGL	unknown	Internal	Linux, Windows
Gazebo	Open Source Robotics Foundation(OSRF)	Active	Apache 2.0	OGRE	ODE/Bullet /Simbody/DART	Internal	Linux, macOS, Windows
MORSE	Academic community	Active	BSD	Blender game engine	Bullet	Blender	Linux, BSD*, macOS
OpenHRP	AIST	Active	Eclipse	Java3D	ODE/Internal	Internal	Linux, Windows
RoboDK	RoboDK	Active	Proprietary	OpenGL	Gravity plugin	Internal	Linux, macOS, Windows, Android, iOS, Debian
SimSpark	O. Obst et al. (+26)	Active	GNU GPL (v2)	Internal	ODE	None	Linux, macOS, Windows
V-Rep	Coppelia Robotics	Active	Proprietary/GNU GPL	Internal	ODE/Bullet/Vortex /Newton	Internal	Linux, macOS, Windows
Webots	Cyberbotics Ltd.	Active	Apache 2.0	Internal (WREN)	Fork of ODE	Internal	Linux, macOS, Windows
4DV-Sim	4D Virtualiz	Active	Proprietary	OGRE	PhysX	Internal	Linux
OpenRAVE	OpenRAVE Community	Active	GNU LGPL	Coin3D/OpenSceneGraph	ODE/Bullet	Internal	Linux, macOS, Windows
Software	Developers	Development status	License	3D rendering engine	Physics engine	3D modeller	Platforms supported

Рисунок 1 – Таблица доступных симуляторов [1].

В современном мире, если у какого-либо продукта, который подразумевает массовое использование отсутствует страница на Wiki, можно смело не доверять этому продукту. Исходя из этого для дальнейшего рассмотрения проходит меньше половины симуляторов: Gazebo, RoboDK, SimSpark, Webots, OpenRAVE. Из оставшихся вариантов, также был исключен RoboDK, так как он распространяется под собственной лицензией, а следовательно продукт является платным, такой вариант нам не подходит, потому что все остальные конкуренты распространяются под бесплатной лицензией Apache 2.0 или GNU GPL. Также был исключен симулятор SimSpark потому, что он является официальным симулятором для проведения RoboCup – это футбол роботов. Известный факт, что если какой-либо продукт приобретает узкую направленность, то для других целей его проблематично будет использовать, ведь задачей стоит создание цифрового двойника, а не игра в футбол.

Таким образом, к дальнейшему рассмотрению остается три симулятора: Gazebo, Webots, OpenRAVE. Перейдем к рассмотрению технических возможностей симуляторов. Для удобства сведем информацию в таблицу 1.

Таблица 1 – Сводная информация о возможностях симуляторов

Название	Основной ЯП	Расширяемость	API для разработки на других ЯП	Поддержка пользователей
Gazebo	C++	Плагины на C++	C++, ROS	e-mail, API документация, форум, пользовательская документация, журнал исправлений, Wiki
Webots	C++	Плагины на C/C++,	C, C++, Python,	Discord канал, e-mail, API

		Собственные API для описания объектов, Свой мета язык для описания объектов - PROTO	Java, Matlab, ROS	документация, форум, пользовательская документация, журнал исправлений, Wiki
OpenRAVE	C++, Python	Плагины на C, C++	C, C++, Python, Matlab	e-mail, API документация, форум, пользовательская документация, журнал исправлений, Wiki

Также необходимо рассмотреть важный аспект – поддерживаемые датчики, сенсоры и приводы. Результаты представлены в таблице 2.

Таблица 2 – Сводная информация о поддерживаемых датчиках

Датчик	Gazebo	Webots	OpenRAVE
Датчик дистанции	+	+	+
GPS	+	+	+
Гироскоп	+	+	+
Камера	+	+	+
Акселерометр	+	+	+
Компас	+	+	+
Лидар	+	+	+
Радар	+	+	+
Датчик положения	+	+	+
Датчик касания	+	+	+
Возможность использовать цифровые двойники датчиков.	Возможность имеется, но вот списка таких датчиков не приводится.	Kinect и лидар сенсоры и датчики расстояния из систем автопилотирования автомобилей.	-
Тормоза	+	+	+
Мотор	+	+	+

Дисплейный модуль	+	+	+
Аудио динамик	+	+	+
Линейный мотор	+	+	+
Мускульный привод	-	+	+
Гусеничный привод	+	+	+

При составлении таблицы 2, были изучены API и пользовательская документация каждого симулятора. Каждый из симуляторов поддерживает необходимый набор датчиков и приводов для создания цифрового двойника, но было замечено, что у симуляторов Gazebo и Webots задокументировано абсолютно все и сама документация подкреплена изображениями и анимацией, а вот симулятор OpenRAVE имеет существенный минус, потому что в течение всего времени выбора симулятора официальный сайт был недоступен и читать документацию приходилось на сторонних ресурсах и абсолютно нормально, что к такой документации доверие отсутствует.

Таким образом остается два кандидата на роль симулятора для реализации проекта: Gazebo и Webots, каждый из которых имеет почти одинаковый функционал и каждый имеет возможность работать с физикой объектов, остается выбрать только по собственным ощущениям и планам на дальнейшее развитие проекта.

Симулятор Gazebo является слишком громоздким и чувствительным к точной настройке, естественно это является несомненным плюсом, но не в нашем случае. Развитие проекта в дальнейшем ляжет на плечи студентов младших курсов, и по планам на это будет выделено не более двух учебных семестров, а этого крайне мало для изучения тонкостей работы с симулятором Gazebo, при условии, что занятия будут проводиться около четырех пар в неделю. В этом плане Webots более прост в понимании студентами, документация написана простым языком, существует своя, упрощенная система создания объектов – PROTO и очень часто автоматическое конфигурирование объектов подходит для корректного использования. А

кросс язычность симулятора Webots очень симпатизирует. Если в Gazebo имеется возможность писать контроллеры для роботов только на языке C++, то в Webots есть возможность программировать на C, C++, Python, Java, Matlab, что позволит сосредоточить свое внимание не на изучении ЯП для написания контроллера, а на так называемой бизнес-логике – логике выполнения команд и поведения. Выбор ЯП таким образом остается на вкус пользователя.

Также важной частью является поддержка пользователей, так как все равно возникнут вопросы, которые не отражены в документации. В этом плане у обоих симуляторов почти одинаковые условия, но Webots отличился наличием Discord канала поддержки, где в одной общем чате присутствуют пользователи и разработчики. Последние в свою очередь стараются максимально подробно ответить на любой интересующий вопрос. Преимуществом чата в сравнении с e-mail общением является то, что в чате видно историю общения с другими пользователями и можно найти ответ на свой вопрос, не тревожа лишний раз разработчиков или ответ поддержки.

Таким образом, финальным симулятором для реализации проекта, который удовлетворяет всем техническим требованиям из пункта 1.2.2, был выбран Webots, потому что он распространяется под бесплатной лицензией, обладает всем необходимым функционалом, имеет низкий порог входа и приятную и человечную поддержку пользователей.

2.3. Выбор языка программирования

В выбранном симуляторе для разработки доступны следующие языки программирования: C, C++, Python, Java, Matlab. Рассмотрим каждый ЯП подробнее.

ЯП C, компилируемый ЯП, который занял нишу в низкоуровневом программировании из-за отсутствия более удобных аналогов. Для разработки контроллера для управления мобильным роботом, где преобладает написание бизнес-логики, ЯП C не очень подходит, потому что его структура и синтаксис заставляет программиста акцентировать внимание на особенностях языка, а не только на бизнес-логике.

ЯП C++ не далеко ушел от ЯП C, поэтому он нам тоже не подходит.

ЯП Python представляет собой интерпретируемый ЯП, который не требует компиляции перед использованием, соответственно, требуется интерпретатор, который присутствует для всех популярных ОС. А вот простота синтаксиса, низкий порог вхождения, большое количество библиотек и высокая популярность в современном сообществе программистов делает этот ЯП отличным кандидатом.

ЯП Java – это компилируемый ЯП, но компиляция происходит в специализированный байт-код, который выполняется на проприетарной Java Virtual Machine. JVM кроссплатформенна, поэтому и код является кроссплатформенным, это несомненный плюс. Также для Java имеется большое количество библиотек, этот ЯП является очень популярным в сообществе, но существенным минусом является высокий порог входа, что нам не совсем подходит, так как проект продолжит свою жизнь как студенческий проект и не у каждого появится желания для работы над проектом еще и изучать новый ЯП. Но не стоит его сбрасывать со счетов,

потому что высокая вероятность того, что студент уже знает этот ЯП и у него будет желание поддерживать проект именно на этом ЯП.

ЯП Matlab этот ЯП обрел популярность в решении задач, где необходимо большое количество технических вычислений. ЯП является частью пакета Matlab, в который входят модули для математических вычислений, визуализации данных и так далее. Несмотря на то, что интерпретатор распространяется под свободной лицензией, для изучения ЯП требуется полный пакет, который уже является платным. ЯП Matlab не совсем подходит для написания контроллера цифрового двойника, так как первоначальная идея этого ЯП – это математические вычисления, а не ЯП для написания бизнес-логики.

Таким образом из пяти первоначальных кандидатов осталось два: Python и Java. Окончательный выбор все равно остановим на ЯП Python, только ради упрощения дальнейшей поддержки проекта, код на Python гораздо понятнее программисту, даже если тот пишет на другом ЯП.

					ВКР-НГТУ-09.03.01-(16-В-2)-016-2020(ПЗ)	Лист
						17
Изм	Лист	№ докум.	Подп.	Дата		

2.4. Выбор среды разработки

Незаменимым атрибутом написания кода является среда разработки. Писать код можно и в обычном текстовом редакторе, или в IDE, встроенной в симулятор, но эти варианты рассмотрены не будут, так как они не имеют таких удобных и уже привычных функций как автоматическое заполнение и подсветка синтаксиса. Поэтому рассмотрим варианты, которые у нас остались.

Python IDLE. Редактор кода, который идет в комплекте с любым интерпретатором. Достаточно мало функционален, мало изменяемый, но зато он идет, что называется «из коробки». Данный вариант нам подходит, но использовать его не удобно из-за невозможности к удобному мультифайловому использованию.

PyCharm. Полноценная IDE, разработанная компанией JetBrains специально для языка Python. Предоставляет удобный редактор кода, с подсветкой синтаксиса и автоматическим дополнением. Возможность отладки и исполнения кода как через интерпретатор, так и через виртуальное окружение. Доступ к редактору пакетов Pip. Удобная работа с иерархией пакетов и директорий проекта. Встроенный редактор для Markdown файлов, для написания документации. Встроенная поддержка систем контроля версий, например Git в связке с GitHub. Работает на всех популярных ОС. Для установки и комфортной работы требуется только инсталлятор.

Sublime Text 3. Редактор текста, с возможностями подсветки синтаксиса. Достаточно популярен в сообществе программистов из-за своей легковесности и широчайших возможностей модификации путем установки плагинов. Удобен для работы с иерархией файлов проекта. Есть возможность вызывать интерпретатор кода для Python. Работает на всех популярных ОС. Для установки требуется только инсталлятор, а вот для комфортной работы

требуется установка ряда плагинов, как минимум Anaconda, который предоставляет удобное использование библиотек.

Visual Studio Code. Редактор кода от Microsoft. Достаточно комфортен для написания кода на любом языке благодаря подсветке синтаксиса и автоматическому дополнению, имеет встроенную поддержку систем контроля версий. Работает на всех популярных ОС. Для установки и комфортной работы требуется не только инсталлятор, но и плагин для работы с Python.

Eclipse + PyDev. Eclipse как IDE изначально была разработана для ЯП Java, но благодаря установке плагина PyDev, данная IDE начинает поддерживать отладку, автоматическое дополнение и подсветку ЯП Python. Работает на всех популярных ОС. Для установки и комфортной работы требуется не только инсталлятор, но и плагин для работы с PyDev.

Каждый из представленных вариантов удобен для использования. В данном же проекте была использована IDE PyCharm из-за того, что она все-таки предоставляет самый большой функционал и изначально создавалась для разработки на Python. По информации с официального сайта JetBrains, при создании IDE, руководствовались пожеланиями сообщества разработчиков, значит можно быть уверенным, что «из коробки» в этой IDE все будет удобно для большинства программистов.

					ВКР-НГТУ-09.03.01-(16-В-2)-016-2020(ПЗ)	Лист
						19
Изм	Лист	№ докум.	Подп.	Дата		

3. Разработка цифрового двойника

3.1. Анализ реального прототипа

Подробные характеристики, аппаратное и программное обеспечение, процесс сборки и другая полезная информация находится на официальной странице разработки робота ELCBot [2].

С данной страницы были выбраны габаритные размеры основных составляющих и сведены в таблицу 3.

Таблица 3 – Габаритные размеры основных составляющих робота

Высота, с учетом ходовой части	1010 мм.
Длина основания	400 мм.
Ширина основания	400 мм.
Масса каркаса с установленным приводом	3 кг.
Клиренс в передней части	18 мм.
Клиренс в задней части	36 мм.
Диаметр ходовых колес	128 мм.
Диаметр заднего колесика	20 мм.
Ширина алюминиевого профиля, из которого составлен каркас	10 мм.
Габаритные размеры корпуса, для крепления ультразвукового датчика. Длина, ширина, высота	40 мм., 20 мм., 10 мм.

Аппаратное обеспечение, которое необходимо реализовать в цифровом двойнике сведено в таблицу 4.

Таблица 4 – Аппаратное обеспечение, необходимое для реализации в цифровом двойнике

Двигатель правого колеса	Привод свеклоподъемного механизма от автомобиля VolksWagen Polo модели 6RO.959.802
Двигатель левого колеса	Привод свеклоподъемного механизма от автомобиля

	VolksWagen Polo модели 6RO.959.801
Инфракрасный датчик и оптический диск на 30 секторов	Данная пара (датчик + диск) образует собой единый блок энкодера, позволяющий подсчитывать количество оборотов колеса
Ультразвуковой датчик расстояния	Таких датчиков на роботе расположено семь штук. Схема расположения представлена на рисунке 2.

Данные таблиц 3 и 4 представляют собой дополнение технических требований 1 и 2 соответственно из пункта 1.2.3.

Фотографии реального робота, без установленных датчиков расстояния представлены на рисунке 3 – вид спереди и рисунке 4 – вид сбоку. На основе данных таблиц 3 и 4, а также рисунков 2, 3 и 4 можно приступить к созданию цифрового двойника.

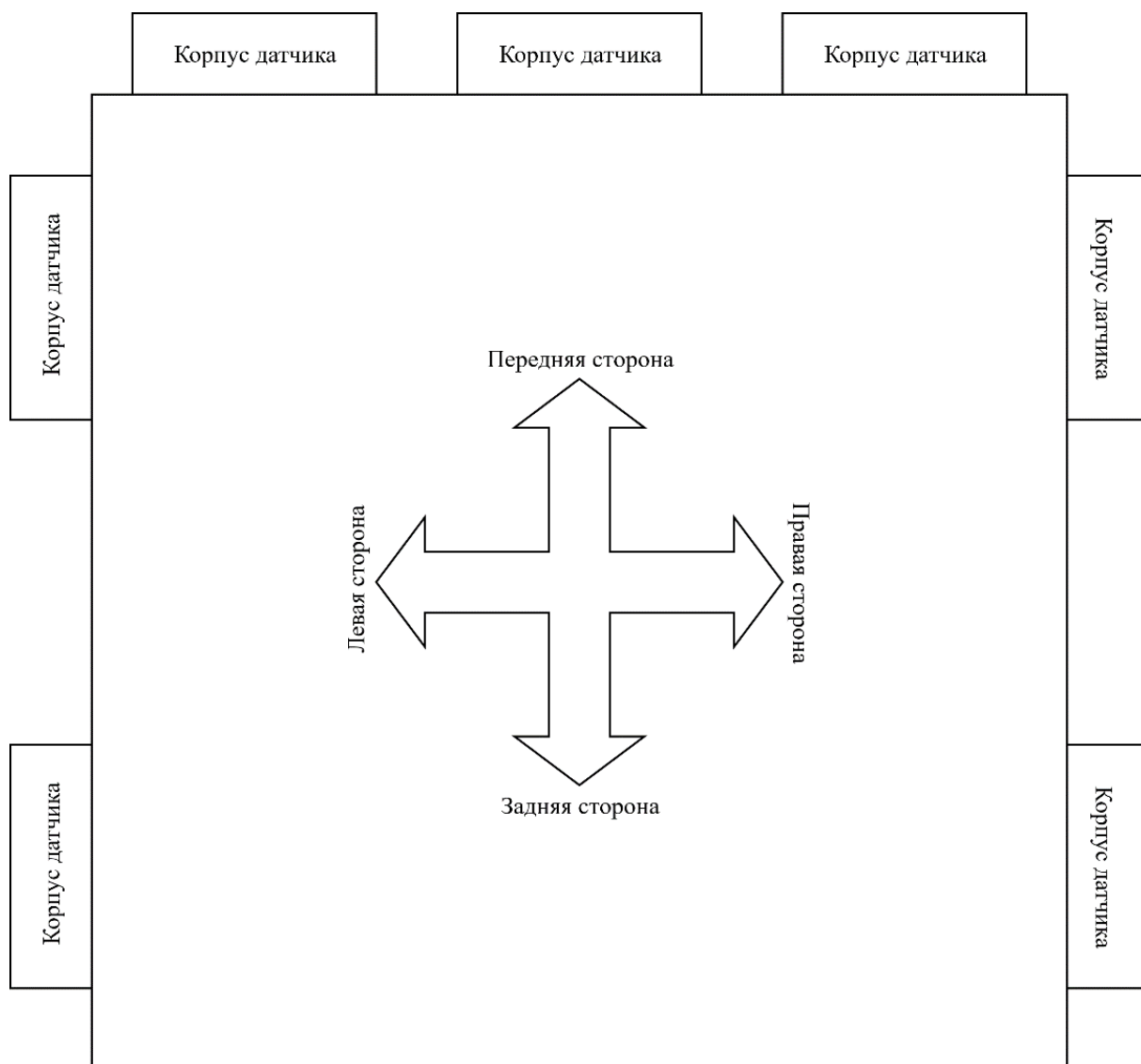


Рисунок 2 – Схема расположения ультразвуковых датчиков.



Рисунок 3 – Фотография робота. Вид спереди

					ВКР-НГТУ-09.03.01-(16-В-2)-016-2020(ПЗ)	Лист
						23
Изм	Лист	№ докум.	Подп.	Дата		



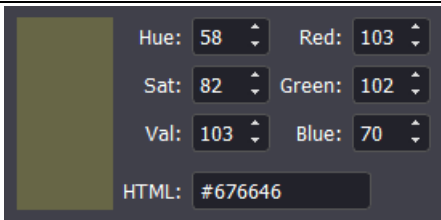
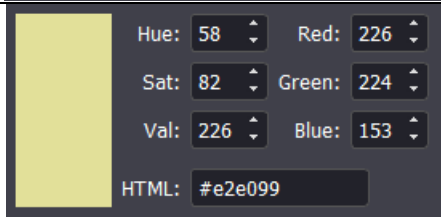
Рисунок 4 – Фотография робота. Вид сбоку.

					ВКР-НГТУ-09.03.01-(16-В-2)-016-2020(ПЗ)	Лист
						24
Изм	Лист	№ докум.	Подп.	Дата		

3.2. Создание окружения цифрового двойника

Цифровой прототип должен находиться в привычном для себя окружении. Поэтому было решено воссоздать часть шестого корпуса НГТУ им Р.Е.Алексеева, крыло, где находятся аудитории 153 и 154. За основу был взята прямоугольная арена, характеристики приведены в таблице 5

Таблица 5 – Характеристики основной арены

Название атрибута	Значение
Размер пола	15 X 10 м.
Высота стен	2 м.
Ширина стен	25 см.
Цвет пола	
Цвет стен	

После этого было создано еще две арены, с размерами пола 10 X 5 м. и 2,5 X 5 м. соответственно, остальные характеристики совпадают с основной ареной. Три арены были расположены таким образом, чтобы представлять собой два коридора с соединяющим их третьим коридором. Итоговый результат можно увидеть на рисунке 5.

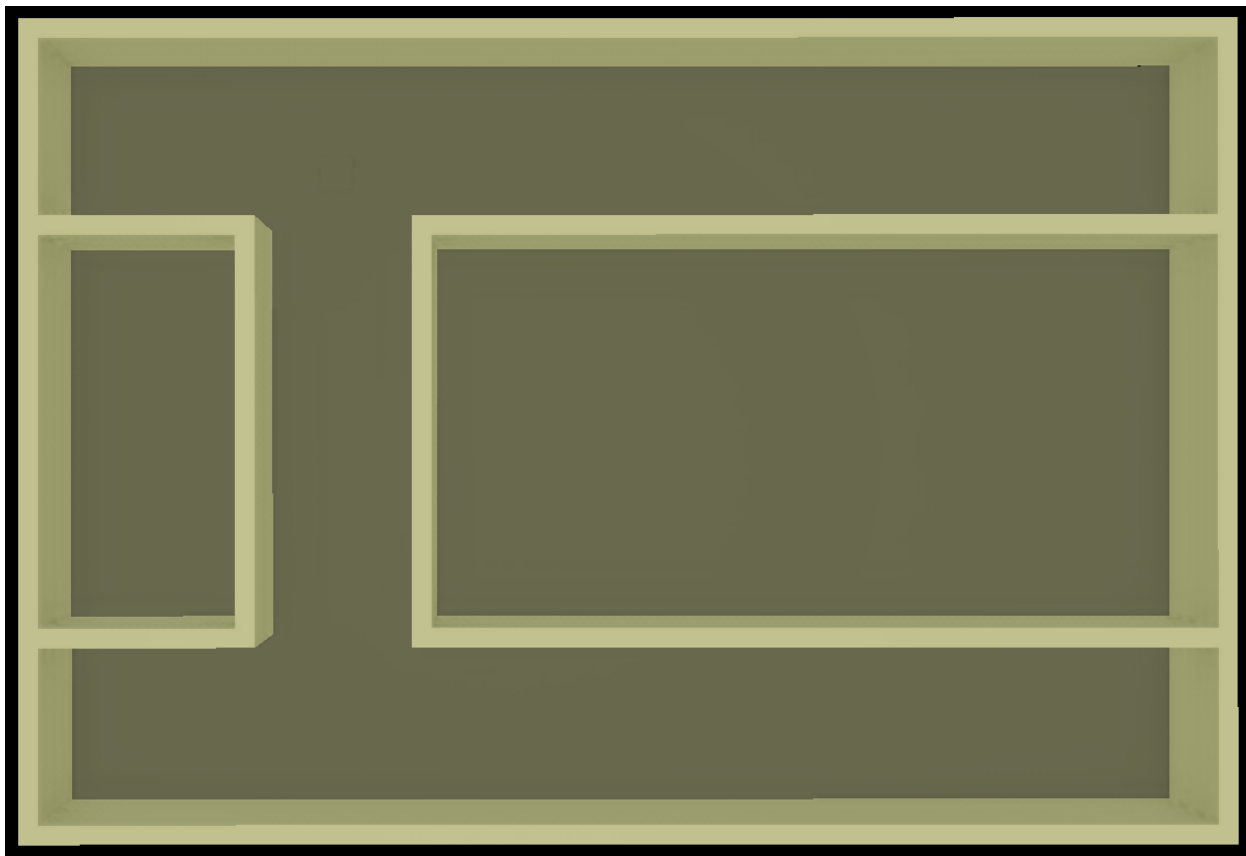


Рисунок 5 – Итоговое окружение робота. Вид сверху.

Можно заметить, что цвет пола и стен немного отличается от представленных в таблице 5, это вызвано тем, что было выбран искусственный источник света и заднее окружение объектов в симулятора под названием «factory», что соответствует освещению на промышленных объектах – ровный, не очень яркий свет, заливающий все помещение. Такая настройка освещения в симуляторе оказалась наиболее подходящей и соответствующей действительности.

После создания окружения можно переходить к созданию цифрового двойника.

3.3. Создание цифрового двойника

Основываясь на данных из пункта 3.1, можно приступить к созданию цифрового двойника. Моделирование в симуляторе осуществляется с помощью узлов [3], у каждого из которых есть список children – узлов наследников, из которых состоит узел.

Начинается все с узла Robot – это абстрактный узел, который отвечает за создание пары робот – контроллер. Для нашего случая поле controller заполняем строкой "ELCController". В список children этого узла необходимо добавить все узлы-компоненты, которые должны быть у цифрового двойника.

Первый компонент, с которого стоит начать – это основание робота. Его размеры 400 X 400 мм. Функция DEF NAME позволяет дать имя узлу, после чего этот узел можно использовать с помощью функции USE NAME. Создадим узел Shape с именем PLATFORM. Зададим ему геометрию и цвет см. рисунок 6.

```
DEF PLATFORM Shape {  
  appearance PBRAppearance {  
    baseColor 0.886275 0.823529 0.886275  
    roughness 1  
    metalness 0  
  }  
  geometry Box {  
    size 0.4 0.005 0.4  
  }  
}
```

Рисунок 6 – Конфигурация узла PLATFORM

Вторым компонентом будет каркас робота. Создадим узел Solid с именем BODY, в списке children узла BODY создадим нужное количество узлов Solid с именем STICK, таким образом мы сможем создать каркас робота, который состоит из планок. По итогу потребовалось 13 объектов STICK Solid. Для примера приведу конфигурирование только одного объекта, см. рисунок

7, потому что остальные используют такие же настройки через функцию USE NAME

```

67 DEF STICK1 Solid {
68   translation 0.328 -0.155 0.147
69   rotation -0.016949705017968317 0.9970414716451949 -0.07497340408030169 -1.38388
70   children [
71     DEF STICK1_SHAPE Shape {
72       appearance DEF STICK_COLOR PBRAppearance {
73         baseColor 0.443137 0.431373 0.447059
74         roughness 1
75         metalness 0
76       }
77       geometry Box {
78         size 0.01 0.01 1
79       }
80     }
81   ]
82   name "stick1"
83   boundingObject USE STICK1_SHAPE
84 }

```

Рисунок 7 – Конфигурация узла STICK1

Рассмотрим рисунок 7 подробнее. Поле translation (строка 68) и rotation (строка 69) отвечают за положение объекта в пространстве, поле geometry (строка 77) отвечает за геометрическую форму объекта, эти поля являются уникальными для каждой планки каркаса, а вот поле appearance (строка 72) можно назвать STICK_COLOR и использовать для каждой планки, чтобы цвет каркаса был одинаковым. Также очень важно поле boundingObject (строка 83), оно отвечает за привязку узла к его физическому представлению, чтобы производить расчет физики, столкновений и взаимодействий с окружающим миром. Поэтому у каждой планки ее форма названа именем (строка 71), чтобы можно было использовать эму же форму для поля boundingObject (строка 83).

В итоге получается каркас робота, см. рисунок 8, который соответствует габаритным размерам оригинала и который можно снаряжать аппаратными средствами, т.е. двигателями, колесами, энкодерами и датчиками расстояния.

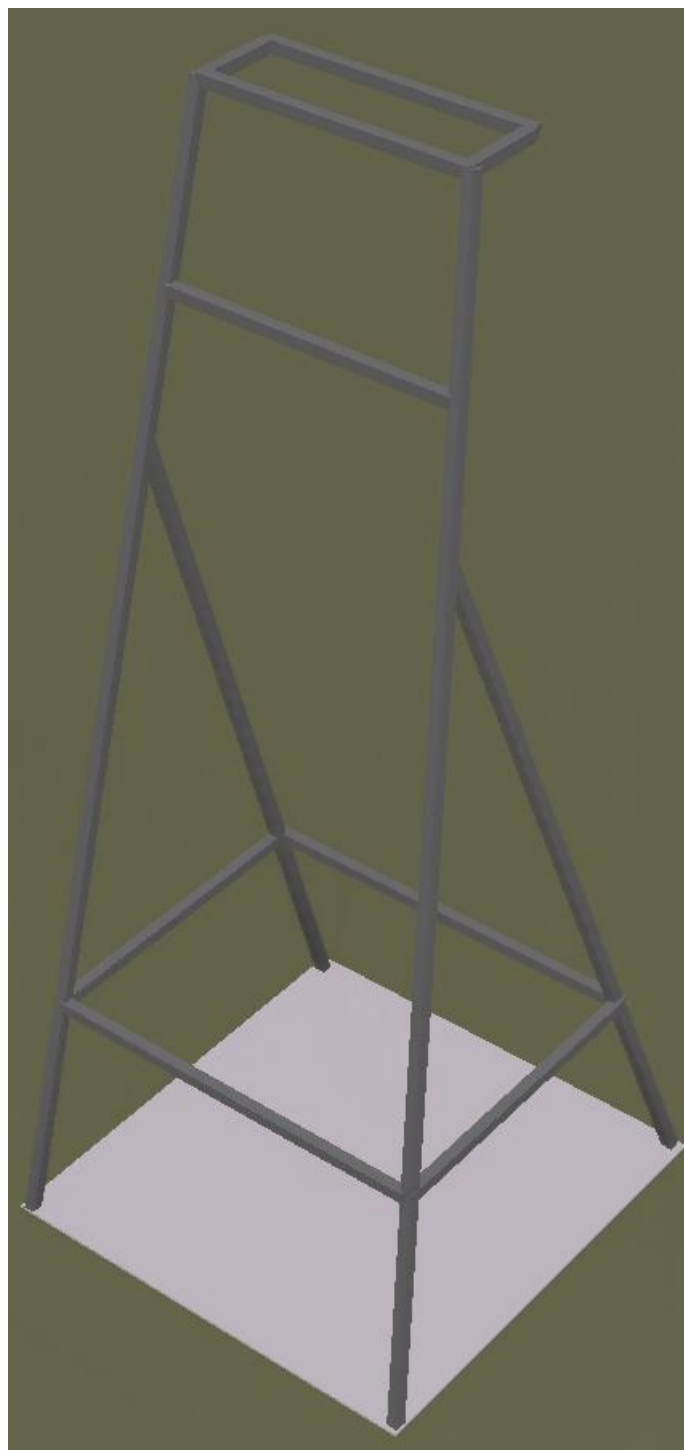


Рисунок 8 – Модель каркаса цифрового двойника.

Первым компонентом аппаратного комплекса будет являться ходовое колесо. Создан будет только один узел ходового колеса, узел второго колеса использует настройки первого, с помощью метода USE NAME. В основу ходового колеса ложится узел HingeJoint, который отвечает вращению поля endpoint вокруг оси, которая описывается в поле jointParameters. Также данный

узел поддерживает список device, в котором указываются функциональные узлы, которые управляются контроллером и отвечают за задающее воздействие на узел. Конфигурация узла HingeJoint для левого колеса представлен на рисунке 9.

```
DEF LEFT_WHEEL HingeJoint {
  jointParameters HingeJointParameters {
    anchor 0.19 0.0200158 0.120003
  }
  device [
    RotationalMotor {
      name "left_wheel_motor"
    }
    PositionSensor {
      name "left_wheel_position_sensor"
    }
  ]
  endPoint Solid {
```

Рисунок 9 – Конфигурация узла HingeJoint для левого колеса

В списке device присутствует узел RotationMotor, которые отвечает за вращение узла HingeJoint и узел PositionSensor, который посчитывает количество оборотов. Таким образом реализованы мотор и энкодер из реального аппаратного обеспечения робота. Поле endpoint представляет собой узел Solid, который необходимо описать как объект, который будет вращаться, для робота это должно быть колесо диаметром 128 мм. За возможность вращения отвечает узел Transform, который указывается в списке children узла Solid. А в самом узле Transform в списке children указывается узел Shape, который описывает форму вращающегося объекта. В нашем случае это будет цилиндр, с радиусом 54 мм. и высотой 20 мм. Финальный вариант сконфигурированного узла Solid для левого колеса представлен на рисунке 10.


```

endPoint Solid {
  translation 0.21 0.0200205 0.120004
  rotation 1 0 0 0
  children [
    DEF RUNNING_WHEEL Transform {
      rotation 0 0 1 -1.5707996938995747
      children [
        Shape {
          appearance PBRAppearance {
            baseColor 0.196078 0.215686 0.215686
            roughness 1
            metalness 0
          }
          geometry Cylinder {
            height 0.02
            radius 0.054
            subdivision 24
          }
        }
      ]
      rotationStep 0
    }
  ]
  name "left_wheel"
  boundingObject USE RUNNING_WHEEL
  physics DEF WHEEL_PHYSICS Physics {
  }
}

```

Рисунок 10 – Узел Solid для левого колеса

Узел Transform был назван RUNNING_WHEEL, чтобы его можно было использовать для поля boundingObject и для конфигурирования правого колеса. Необходимым условием для корректного расчета физики является недопустимость нахождения якоря внутри объекта вращения, поэтому поле anchor в узле jointParameters (рисунок 9) отличается от поля translation в узле endPoint (рисунок 10) на 20 мм. по оси x.

Конфигурация правого колеса отличается только положением полей anchor и endPoint, другие поля используют функцию USE NAME. Итоговая конфигурация представлена на рисунке 11.

					ВКР-НГТУ-09.03.01-(16-В-2)-016-2020(ПЗ)	Лист
						31
Изм	Лист	№ докум.	Подп.	Дата		

```

DEF RIGHT_WHEEL HingeJoint {
  jointParameters HingeJointParameters {
    anchor -0.19 0.0200163 0.120003
  }
  device [
    RotationalMotor {
      name "right_wheel_motor"
    }
    PositionSensor {
      name "right_wheel_position_sensor"
    }
  ]
  endPoint Solid {
    translation -0.21 0.0200213 0.120004
    rotation 1 0 0
    children [
      USE RUNNING_WHEEL
    ]
    name "RIGHT_WHEEL"
    boundingObject USE RUNNING_WHEEL
    physics USE WHEEL_PHYSICS
  }
}

```

Рисунок 11 – Конфигурация узла HingeJoint для правого колеса

Правое и левое колеса расположены таким образом, чтобы клиренс в передней части составлял 18 мм.

Следующим компонентом аппаратной части является пассивное заднее колесико, которое имеет две степени свободы, в отличие от ходовых колес. Для упрощения процесса моделирования было решено заменить колесико на сферу и использовать узел Hinge2Joint, который позволяет вращать endPoint по двум осям. Сфера создавалась узлом Sphere, радиус сферы 17 мм. добавляем 2 мм. отступ от основания каркаса и получаем клиренс задней части 36 мм. Финальная конфигурация узла Hinge2Joint для заднего колесика представлена на рисунке 12.

```

DEF BACK_WHEEL_JOINT Hinge2Joint {
  jointParameters HingeJointParameters {
    anchor 0.01 -0.02 -0.189997
  }
  endPoint Solid {
    translation 0.01 -0.02 -0.189997
    children [
      DEF BACK_WHEEL Transform {
        children [
          Shape {
            appearance PBRAppearance {
              baseColor 0.196078 0.215686 0.215686
              roughness 1
              metalness 0
            }
            geometry Sphere {
              radius 0.017
              subdivision 5
              ico FALSE
            }
          }
        ]
      }
    ]
  }
  name "BACK_WHEEL"
  boundingObject USE BACK_WHEEL
  physics Physics {
  }
}

```

Рисунок 12 - Конфигурация узла Hinge2Joint для заднего колесика

Предпоследней частью аппаратного комплекса являются датчики расстояния. Они объявляются узлом DistanceSensor, в список children помещается узел Shape, который отвечает за геометрию датчика. Для имен датчиков была разработана система именования в зависимости от положения датчика на каркасе робота. Всего на каркасе расположено 7 датчиков список именований приведен в таблице 6. Смотреть на модель необходимо так же, как на рисунке 2. DS в контексте таблицы Distance Sensor

					ВКР-НГТУ-09.03.01-(16-В-2)-016-2020(ПЗ)	Лист
						33
Изм	Лист	№ докум.	Подп.	Дата		

Таблица 6 – Соответствия имен датчиков с их положением

Название датчика	Описание
DS_FC	Front Center – передний центральный датчик
DS_FL	Front Left – передний левый датчик
DS_FR	Front Right – передний правый датчик
DS_LF	Left Front – левый передний датчик
DS_LB	Left Back – левый задний датчик
DS_RF	Right Front – правый передний датчик
DS_RB	Right Back – правый задний датчик

Узел Shape для всех датчиков одинаков, называется DS_SENSOR_SHAPE и его габаритные размеры равны 10 X 20 X 40 мм.

На рисунке 13 приведено описание переднего центрального датчика, где и сконфигурирован узел DS_SENSOR_SHAPE. Поле lookupTable отвечает за минимальные и максимальные показания датчика. Данное поле заполнено таким образом, чтобы минимальное значение было 0, а максимальное 255, именно такие значения может принимать датчик расстояния на реальном датчике.

```

DEF DS_FC DistanceSensor {
  translation -5.38735e-07 0.21 0.16
  rotation 0 1 0 -1.5707996938995747
  children [
    DEF DS_SENSOR_SHAPE Shape {
      appearance PBRAppearance {
        roughness 1
        metalness 0
      }
      geometry Box {
        size 0.01 0.02 0.04
      }
    }
  ]
  name "ds_fc"
  lookupTable [
    0 0 0
    0 255 0
  ]
}

```

Рисунок 13 – Конфигурация переднего центрального датчика.

Последним узлом будет добавлена камера, которая заменяет блок Kinect на реальном роботе. В качестве тела выбран узел Box черного цвета с размерами 18 X 7 X 4 см. Разрешение изображения с камеры 720 X 480 пикселей. Конфигурация узла представлена на рисунке 14.

```

Camera {
  translation -1.46928e-07 0.995 -0.025
  rotation 0 1 0 -3.1415853071795863
  children [
    Transform {
      rotation 1 0 0 1.57
      children [
        Shape {
          appearance PBRAppearance {
            baseColor 0 0 0
            roughness 1
            metalness 0
          }
          geometry Box {
            size 0.18 0.07 0.04
          }
        }
      ]
    }
  ]
  width 720
  height 480
}

```

Рисунок 14 – Конфигурация камеры.

Дополнительным узлом аппаратной части робота является узел Receiver, который обеспечивает возможность принимать команды от узла Emiteer. Через узел Receiver в дальнейшем реализовано управление роботом при помощи клавиатуры. В реальном роботе, данный модуль представляет собой Wi-Fi соединение с локальной сетью, по которой на робота можно передавать управляющие команды. Данный параметр не был учтен в пункте 3.1, потому что Wi-Fi соединение обеспечивает Raspberry PI B+, реализация которой не подразумевается в цифровом двойнике.

В итоге, смоделировав каркас робота, разместив на нем ходовые колеса, заднее колесико и семь датчиков расстояния получаем готовый цифровой двойник. На рисунке 15, который соответствует рисунку 3 и на рисунке 16,

который соответствует рисунку 4, можно увидеть изображение получившегося цифрового двойника.

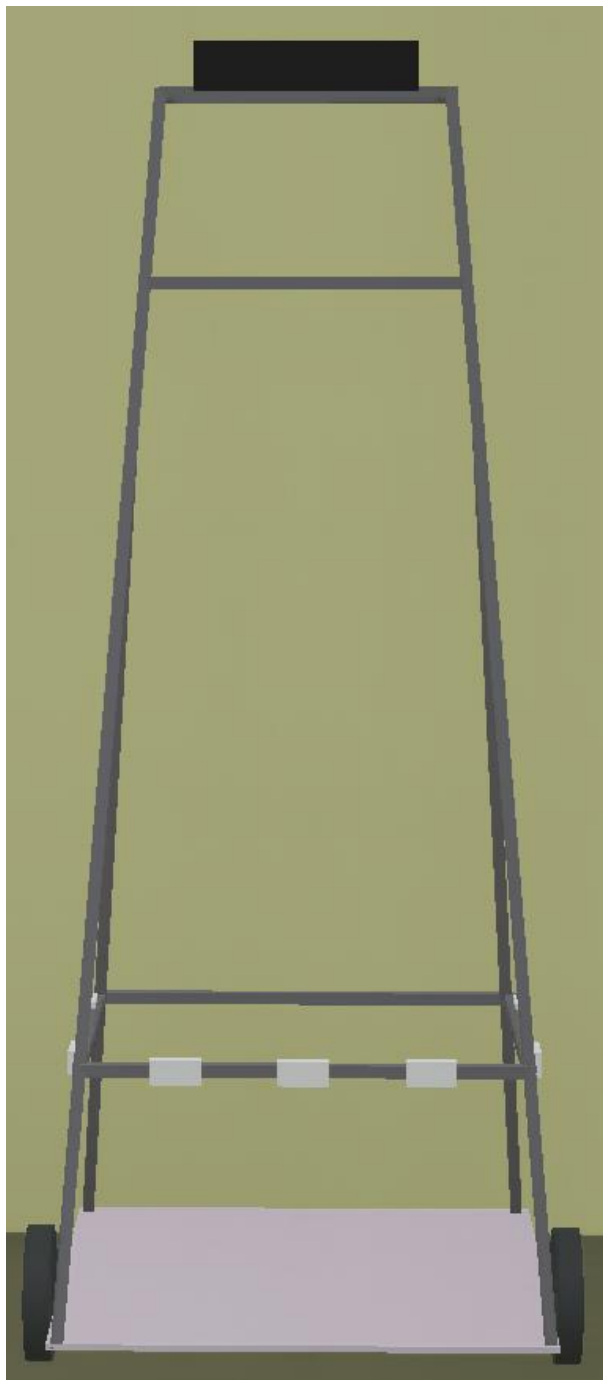


Рисунок 14 – Изображение цифрового двойника. Вид спереди

					ВКР-НГТУ-09.03.01-(16-В-2)-016-2020(ПЗ)	Лист
						37
Изм	Лист	№ докум.	Подп.	Дата		

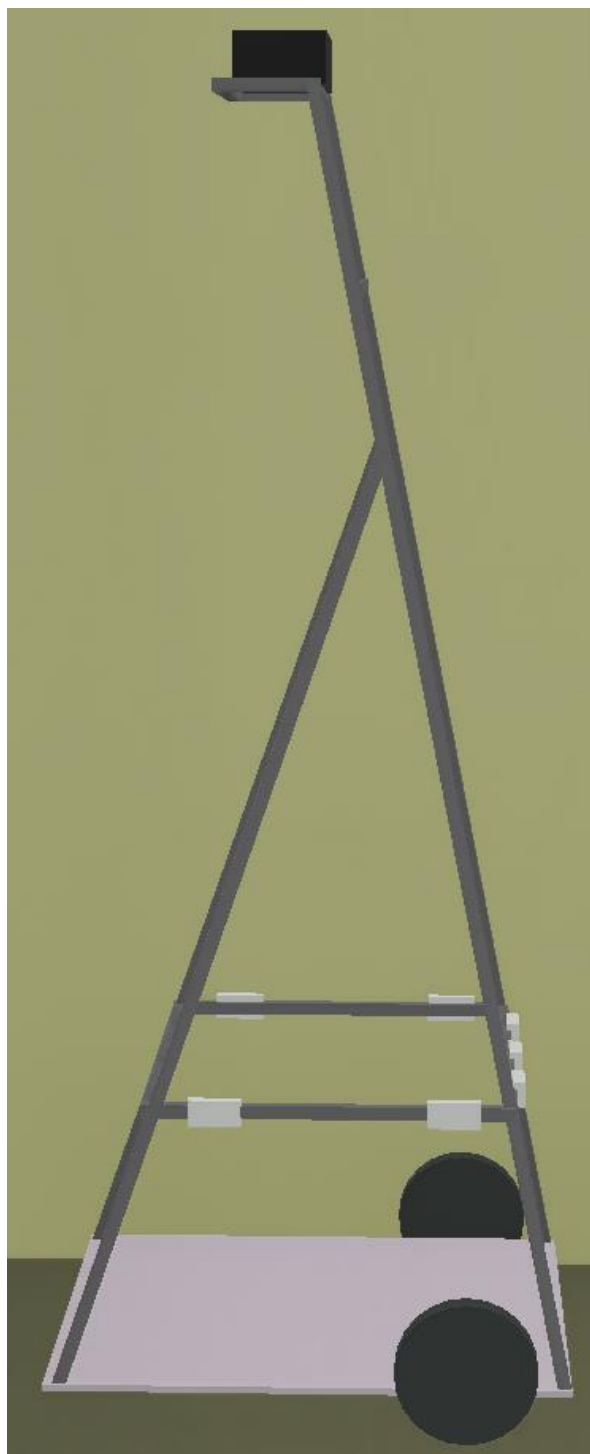


Рисунок 15 - Изображение цифрового двойника. Вид сбоку

					ВКР-НГТУ-09.03.01-(16-В-2)-016-2020(ПЗ)	Лист
						38
Изм	Лист	№ докум.	Подп.	Дата		

4. Разработка системы управления движением

На рисунке 16 представлена общая схема управления движением, которую должен поддерживать цифровой двойник.

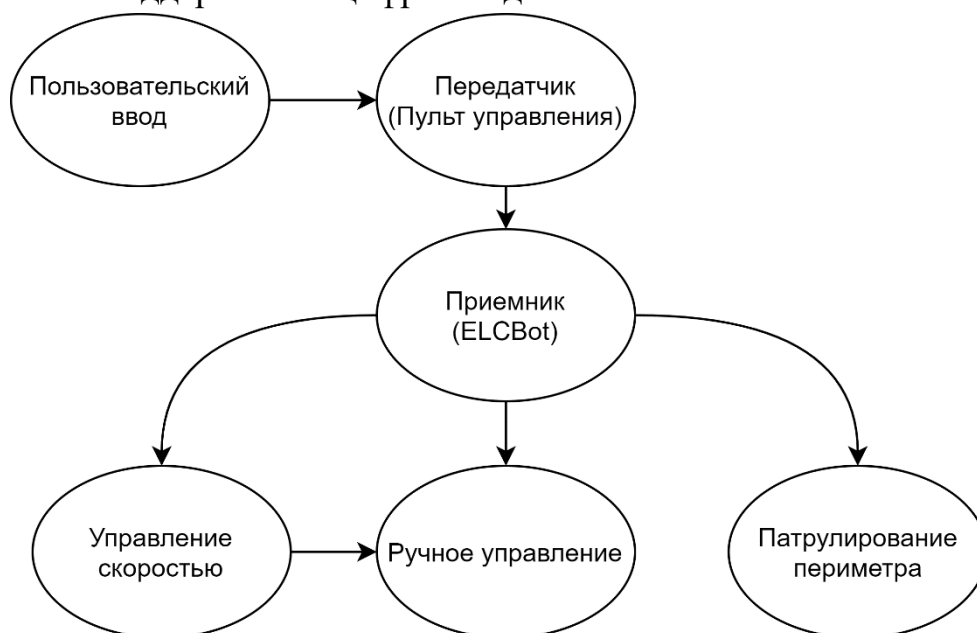


Рисунок 16 - Общая схема управления движением

4.1. Декомпозиция объектов и написание ручного управления

Главным объектом программы будет цифровой двойник. Поэтому создадим класс ELCBot, который наследуется от класса Robot, в котором будут храниться настройки и аппаратное обеспечение робота. В конструкторе при помощи API симулятора инициализируем моторы, энкодеры, датчики расстояния, камеру и приемник команд.

Следующим шагом будет разбор необходимых функций движения, которые формулируются исходя из окружения робота (6). Список функций и их описание приведены в таблице 7.

Таблица 7 – Описание основных функций

Название	Имя	Описание	Возвращаемое значение
Движение вперед	move_forward	Подача одинаковой	-

		Положительной скорости на оба колеса	
Движение назад	move_backward	Подача одинаковой отрицательной скорости на оба колеса	-
Остановка	stop	Подача нулевой скорости на оба колеса	-
Разворот налево	turn_left	Подача на левое колесо отрицательной скорости, а на правое положительной. Скорость одинаковой величины.	-
Разворот направо	turn_right	Подача на левое колесо положительной скорости, а на правое отрицательной. Скорость одинаковой величины.	-
Детектирование препятствия спереди	front_obstacle	Проверка, что значения с передних датчиков меньше константной величины	Булевоe
Детектирование препятствия слева	left_obstacle	Проверка, что значения левых боковых датчиков меньше константного значения	Булевоe

Детектирование препятствия справа	right_obstacle	Проверка, что значения правых боковых датчиков меньше константного значения	Булевое
Детектирование параллельности объекту слева	left_equal	Проверка, что модуль разности показаний левых боковых датчиков меньше константного отклонения	Булевое
Детектирование параллельности объекту справа	right_equal	Проверка, что модуль разности показаний правых боковых датчиков меньше константного отклонения	Булевое
Детектирование окончания стены	end_wall	Проверка, что значения с переднего бокового датчика больше максимально допустимого, а с заднего бокового меньше или равно минимальному	0, 1, -1
Регулировка скорости движения	speed_handler	Проверка сигнала с передатчика и установка скорости от 1 до 10	-

На рисунке 17 представлены клавиши управления и функции которые они выполняют при нажатии.

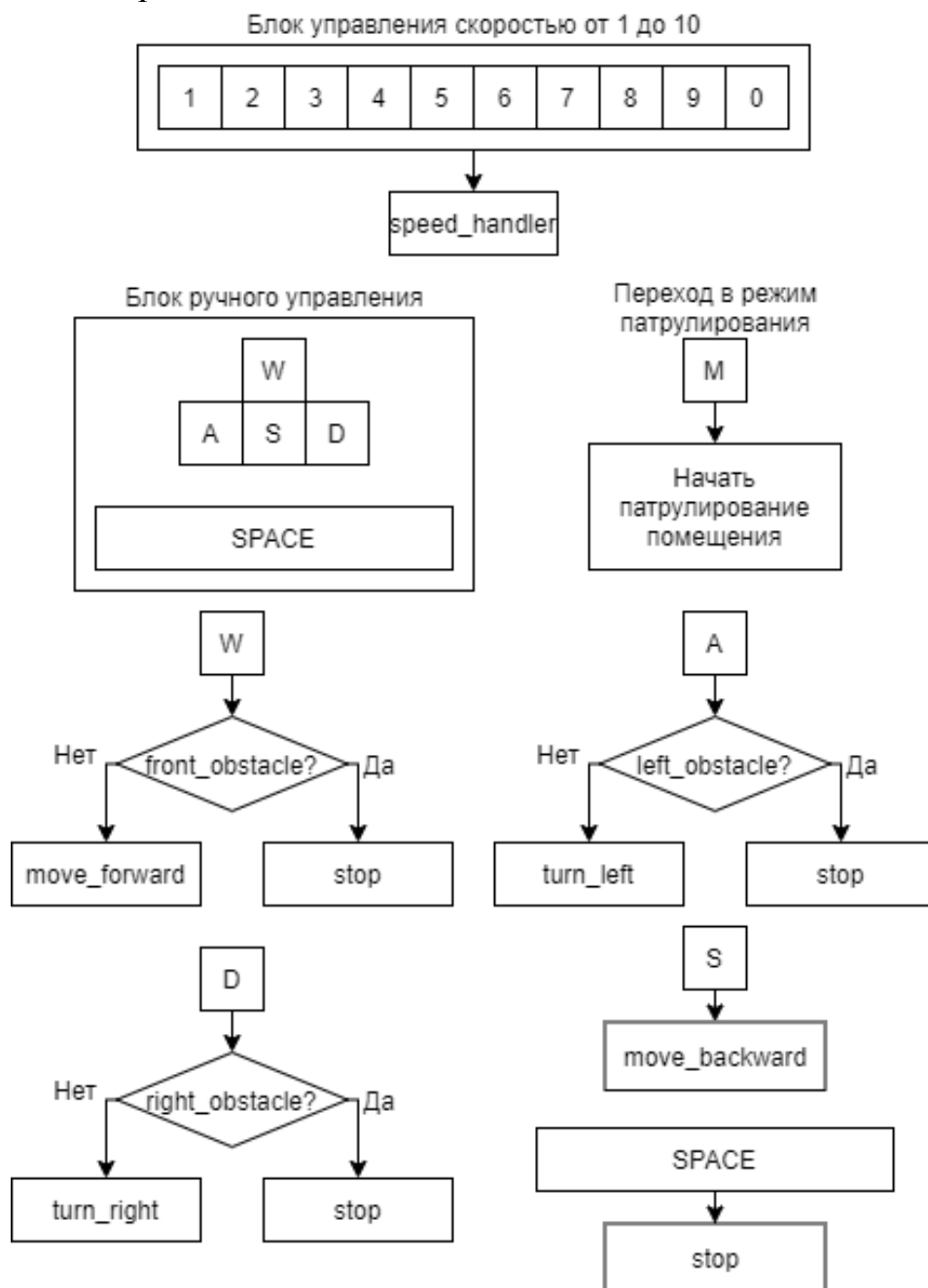


Рисунок 17 – Назначения клавиш

После реализации в коде данного функционала, было проведено тестирование и эмпирическим путем подобраны константы для датчиков расстояний. 50 – для передних датчиков, 40 – для боковых. Именно эти значения позволяют роботу безопасно разворачиваться и не сталкиваться с препятствиями.

4.2. Режим патрулирования

Исходя из окружения робота – часть здания, можно декомпозироваться ряд задач, которые должен выполнять робот при патрулировании периметра [6]:

- Детектирование препятствий
- Вход в поворот при внутреннем угле
- Вход в поворот при внешнем угле
- Ровное движение вдоль стены

На рисунке 18 и 19 представлено две части блок-схемы алгоритма автоматического движения. На рисунке 18 – часть, когда движение вперед заблокировано, на рисунке 19, когда разблокировано. Данный алгоритм выполняет все задачи, которые были поставлены перед роботом при патрулировании периметра. Если описывать словами, то получается, что робот едет вперед, пока не встретит препятствие. После обнаружения препятствия он попытается встать параллельно ему (расстояние между передними датчиками и боковыми одинаковое), затем он начнет движение вдоль этого объекта, будь то стена или человек, при помощи PID регулятора и алгоритма подбора коэффициентов в зависимости от рельефа объекта или его окончания.

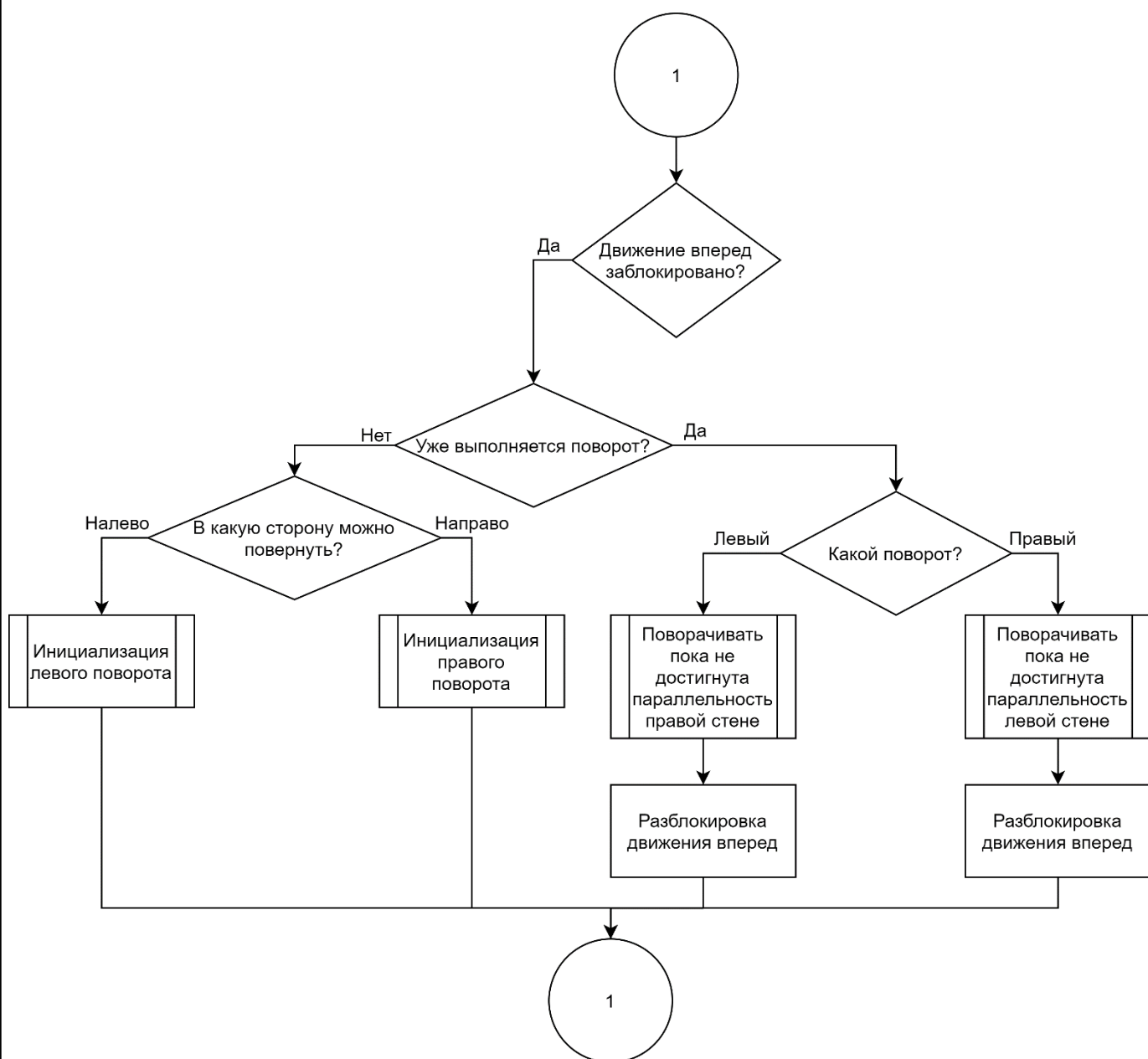


Рисунок 18 – Блок-схема алгоритма движения при заблокированном движении вперед

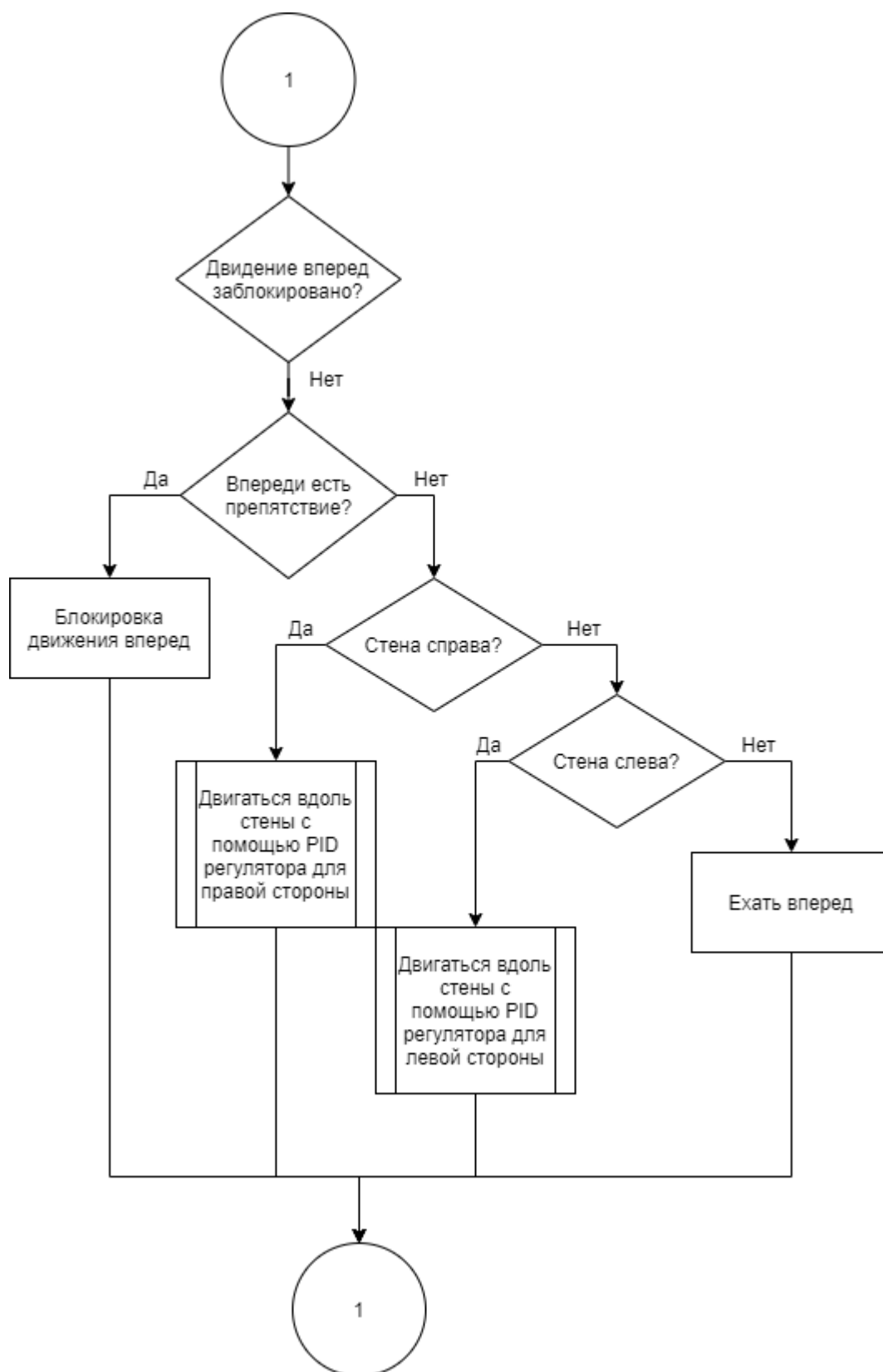


Рисунок 19 - Блок-схема алгоритма движения при разблокированном движении вперед

4.3. PID регулятор

4.3.1. Алгоритм движения вдоль стены

Одной из важных частей алгоритма движения является ровное движение вдоль стены. Для реализации этой части был использован PID регулятор - пропорционально-интегрально-дифференциальный регулятор выдает на выходе сигнал для плавного управления объектом управления при помощи управляющего устройства, с детектированием ошибок, при помощи обратной связи.[4] В нашем случае мы управляем моторами при помощи алгоритма движения, детектируя ошибки при помощи бокового переднего датчика. Только передний боковой датчик выбран не случайно. Во время тестирования работы PID регулятора, были испробованы различные комбинации обратной связи:

- Оба боковых датчика
- Только передний
- Только задний
- Только передний с расположением на одной оси с двигателями

Последний вариант с передним боковым датчиком на одной оси с двигателями с максимальной точностью позволяет отслеживать неровность стен и корректировать движение робота. На реальном прототипе передний датчик не находится на одной оси с двигателями. Именно благодаря цифровому двойнику теперь можно исправить это с учетом минимальных трудозатрат на тестирование.

Подробно останавливаться на математическом описании PID регулятора не будем. Важно знать, что результатом работы являются гармонические затухающие колебания и что у него есть три составляющих [5]:

- Пропорциональная $u_p = k_p * e(t)$, где $e(t)$ – функция ошибки. При итерационном расчете это разность ошибки на текущем шаге и на предыдущем.
- Интегральная $u_i = k_i * \int e(t)dt$, где $e(t)$ – функция ошибки. При итерационном расчете интеграл равен накопленной ошибке.
- Дифференциальная $u_d = k_d (y(t) - y(t-1))$, где $y(t)$ – функция обратной связи. Для нас это показания с датчиков на текущем шаге и на предыдущем.

Результирующей функцией будет сумма всех трех составляющих: $u_{pid} = u_p + u_i + u_d$. Самой сложной частью всегда является настройка коэффициентов, потому что необходимо достичь колебаний, которые затухнут за наименьшее количество периодов при минимальной амплитуде и будут так же быстро реагировать на резко изменившиеся условия. Подбор коэффициентов подробно рассмотрен в пункте 4.3.2.

На рисунке 20 представлена блок-схема алгоритма движения вдоль стеры при помощи PID регулятора. Если описывать словами, получается, что высчитываются составляющие PID регулятора и с помощью выходного сигнала регулятора, сумма составляющих, корректируется скорость двигателей, из-за чего робот едет по определенной траектории, если же был обнаружен конец стены, происходит изменение коэффициентов, чтобы робот вошел в резкий поворот за угол и после того, как он снова будет ехать вдоль стены, коэффициенты возвращаются в начальные значения. Решение о изменении коэффициентов при окончании стены было принято в ходе тестирования, потому что настроить их универсально не представляется возможным и необходимо подбирать коэффициенты для каждого не типичного случая при выполнении алгоритма.

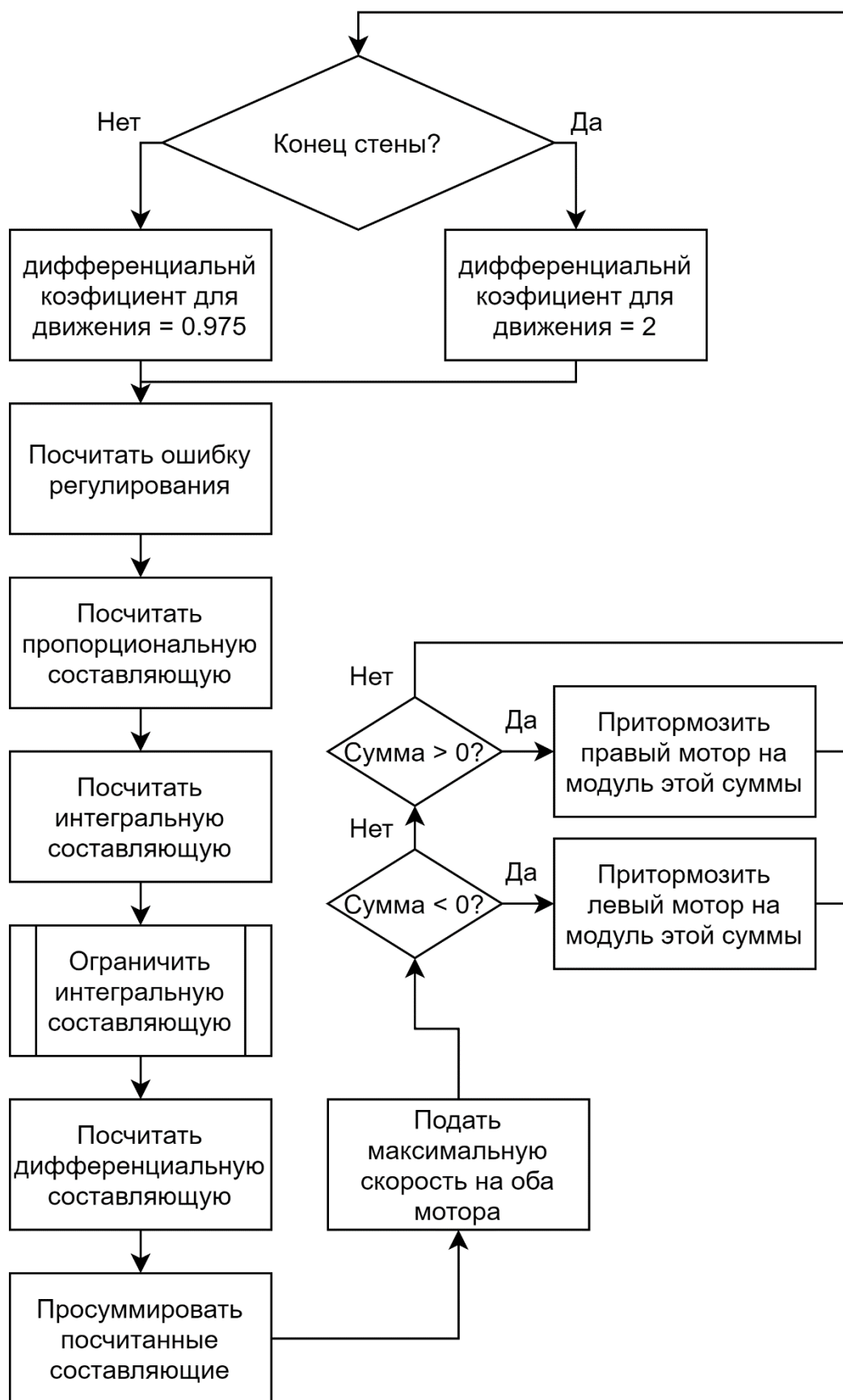


Рисунок 20 – Алгоритм движения вдоль стены при помощи PID регулятора

4.3.2. Настройка коэффициентов PID регулятора

Существует несколько способов настройки [3]:

- Теоретическая – создать математическую модель системы, с учетом всех нюансов, которые могут возникнуть. И на данной системе итерационно подбирать коэффициенты таким образом, чтобы задача решалась максимально эффективно.
- Инженерная – создание отладочного стенда, и по данным с графиков и осциллограммам отклика системы на определенное воздействие регулировать и подбирать коэффициенты эмпирическим путем.
- Реальная – эмпирический подбор коэффициентов с натурными испытаниями на реальном прототипе.

Симулятор предоставляет построение графиков для данных с датчиков, а вот сам цифровой двойник хоть и можно назвать отладочным стендом, но для получения точных осциллограмм он не предназначен. Поэтому был выбран гибридным способ. Реально-инженерный. Робот двигался в своем виртуальном окружении и на основе его движения и данных с датчиков расстояния подбирались коэффициенты.

Изначально все коэффициенты равно 0.

Сначала настраивается пропорциональный коэффициент. На цифровой модели он отвечает за поворот во время стабильности данных с графиков. Настройка производится в пределах от 0 до 1 таким образом, чтобы во время начала движения модель не разворачивалась на месте, а начинала плавное движение вперед с небольшим углом поворота. В данном случае пропорциональный коэффициент равен 0.1

Затем при настроенном пропорциональном коэффициенте происходит настройка интегрального коэффициента, он отвечает за период колебаний системы. То есть как часто робот будет совершать колебания, прежде чем выровняется. Данный коэффициент находится уже в диапазоне от 0.01 до 0.001 для данной цифровой модели. Эмпирическим путем был подобран интегральный коэффициент равный 0.022

В последнюю очередь подбирается дифференциальный коэффициент, на цифровой модели он отвечает за амплитуду колебаний. На рисунке 21 и 22 видно, как сильно отличается амплитуда при разнице всего в 0.025. Данный коэффициент для данной системы лежит в диапазоне от 0 до 3 и финальный вариант равен 0.975.

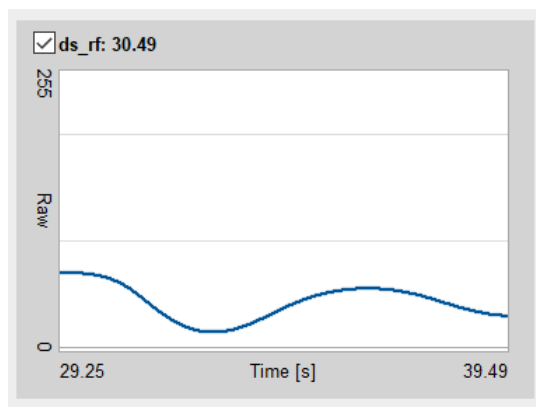


Рисунок 21 – Данные с датчика при дифференциальном коэффициенте равном 1

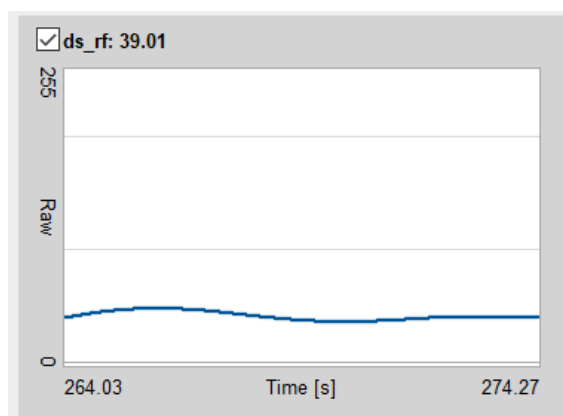


Рисунок 22 – Данные с датчика при дифференциальном коэффициенте равном 0.975

После подбора всех коэффициентов модель смогла двигаться вдоль стены, но слабо и медленно реагировала на момент, когда теряла стену. Возникла необходимость в подборе коэффициентов для PID регулятор для такого случая. Эмпирическим путем было выявлено, что дифференциальный коэффициент оказывает максимальное влияние на модель при его значении равном 2. На рисунке 23 видно, что регулятор пытается поддерживать оптимальное расстояние до стены при прохождении в поворот. Прерывистость графика в начале говорит о том, что модель потеряла стену, но в итоге вернулась в первоначальное состояние, но амплитуда слишком высокая и после возврата необходимо опять снизить дифференциальный коэффициент до 0.975.

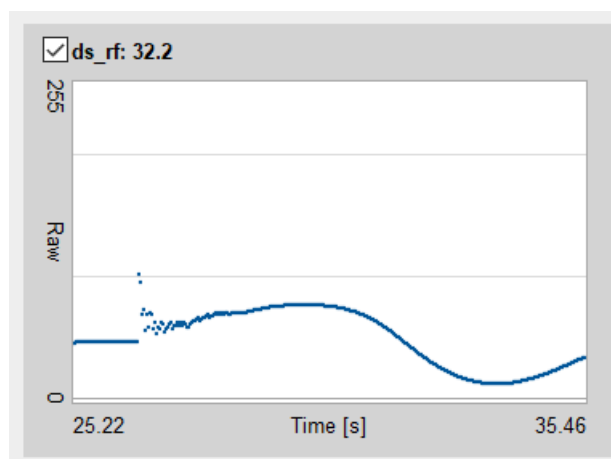


Рисунок 23 - Данные с датчика при дифференциальном коэффициенте равном 2 при проходе в поворот

5. Тестирование системы

Тестирование системы проводилось в окружении робота, для эксперимента даже расставлены коробки, которые символизируют скопления людей. Условия испытаний представлены на рисунке 24. Для успешного теста цифровой двойник должен объехать на максимальной скорости весь периметр, объехав все препятствия. Он успешно с этим справился за 3 минуты и 57 секунд. Помещение без препятствий он патрулирует за 3 минуты ровно.

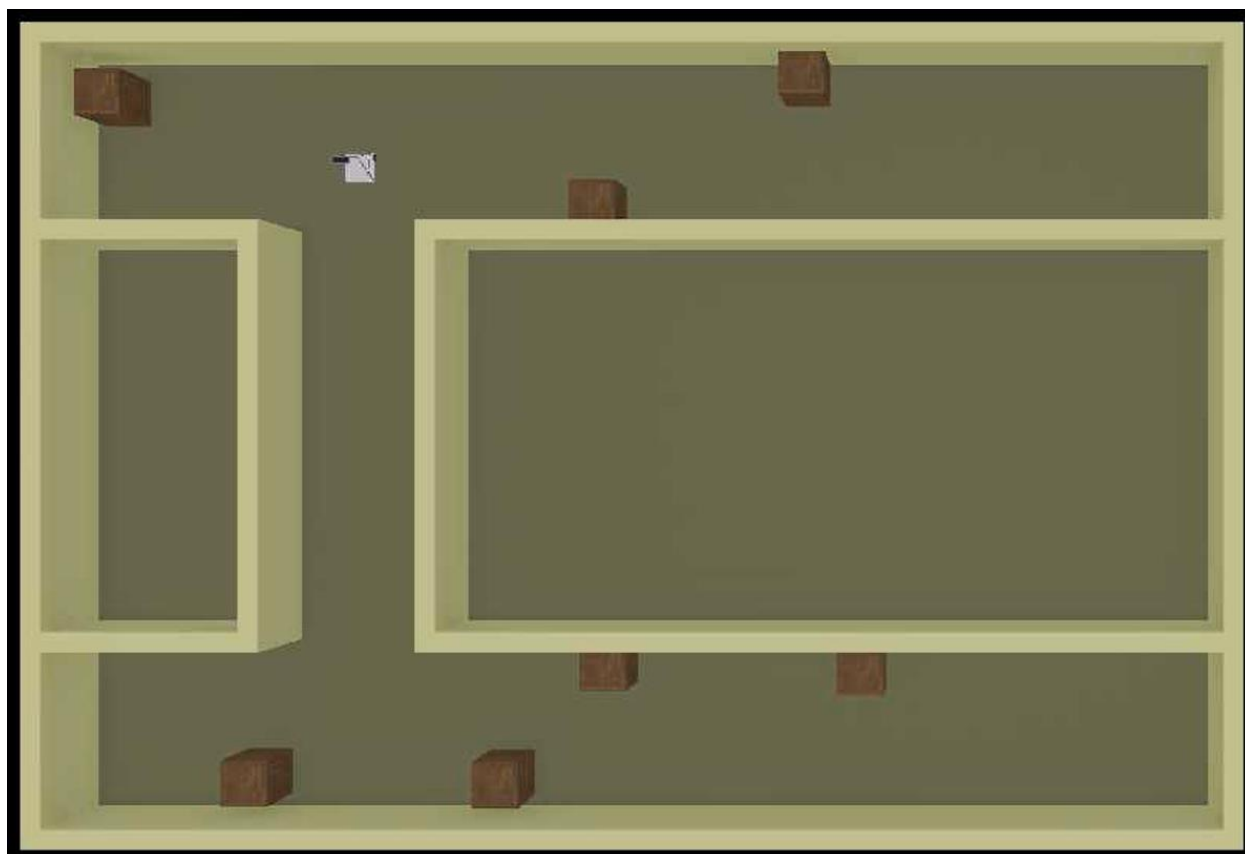


Рисунок 24 – Тестирование цифрового двойника

Заключение

В ходе выполнения работы был создан цифровой двойник реального прототипа робота ELCBot. Точность цифрового двойника достаточно высокая и с его помощью можно производить тестирование различных алгоритмов перед применением их на реальном прототипе.

Для цифрового двойника было создано виртуальное окружение, представляющее собой часть 6 корпуса НГТУ им. Р.Е. Алексеева.

Также был написан алгоритм патрулирования помещения с обходом препятствий. В основу алгоритма лег PID регулятор для движения двойника параллельно стене и для вхождения во внешний поворот за угол.

Данный алгоритм был успешно протестирован на цифровом двойнике в виртуальном окружении будущего использования реального прототипа

В дальнейшем можно производить перенос данного алгоритма на реального робота.

					ВКР-НГТУ-09.03.01-(16-В-2)-016-2020(ПЗ)	Лист
						53
Изм	Лист	№ докум.	Подп.	Дата		

Перечень сокращений

ОС – операционная система

GUI – графический пользовательский интерфейс

ЯП – язык программирования

IoT – Internet of things (Интернет Вещей)

Бизнес-логика – логика, описывающая поведение объектов предметной области, а не логику написания алгоритмов на определенном ЯП.

IDE – интегрированная среда разработки

PID регулятор – Пропорционально интегрально дифференциальный регулятор.

					ВКР-НГТУ-09.03.01-(16-В-2)-016-2020(ПЗ)	Лист
						54
Изм	Лист	№ докум.	Подп.	Дата		

Список литературы

1. Robotics simulator // Wikipedia URL:
https://en.wikipedia.org/wiki/Robotics_simulator
2. Проект робота телеприсутствия ELCBot // Github URL:
<https://tinyroboticsteam.github.io/ElcBot/>
3. Webots Reference Manual // Webots URL:
<https://www.cyberbotics.com/doc/reference/nodes-and-api-functions>
4. Манфред Шляйхер. Техника автоматического регулирования для практиков – Фульда, 2006 – 124 с.
5. Карпов В. Э. ПИД-управление в нестрогом изложении – Москва, 2012 – 34 с.
6. Роберт Мартин. Чистый код. Создание, анализ, рефакторинг – СПб, издательство Питер, 2019 – 464 с.

Приложения

Приложение А. Код мира симуляции

```
#VRML_SIM R2020a utf8
WorldInfo {
}
Viewpoint {
  orientation 1 0 0 4.71238898038469
  position 4.257669365809002 33.39223912447635 -0.0680736907235865
  follow "elc-bot"
}
TexturedBackground {
  texture "factory"
  skybox FALSE
}
TexturedBackgroundLight {
  texture "factory"
  castShadows FALSE
}
RectangleArena {
  floorSize 15 10
  floorAppearance DEF FLOOR_COLOR PBRAppearance {
    baseColor 0.403922 0.4 0.27451
    roughness 1
    metalness 0
  }
  wallThickness 0.25
  wallHeight 2
  wallAppearance DEF WALL_COLOR PBRAppearance {
    baseColor 0.886275 0.878431 0.6
    roughness 1
    metalness 0
  }
}
RectangleArena {
  translation 2.5 0 0
  name "rectangle arena(1)"
  floorSize 10 5
  floorAppearance USE FLOOR_COLOR
  wallThickness 0.25
  wallHeight 2
  wallAppearance USE WALL_COLOR
}
RectangleArena {
  translation -6.25 0 0
  name "rectangle arena(2)"
  floorSize 2.5 5
  floorAppearance USE FLOOR_COLOR
  wallThickness 0.25
  wallHeight 2
  wallAppearance USE WALL_COLOR
}
Robot {
  translation -3.95057 0.05 -3.08414
  rotation 0 1 0 3.14157
  children [
    DEF PLATFORM Shape {
```

```

    appearance PBRAppearance {
        baseColor 0.886275 0.823529 0.886275
        roughness 1
        metalness 0
    }
    geometry Box {
        size 0.4 0.005 0.4
    }
}
DEF BODY Solid {
    translation 2.57125e-07 0.160006 -0.049997
    rotation 0 0 1 1.5708
    children [
        DEF STICK1 Solid {
            translation 0.328 -0.155 0.147
            rotation -0.016949705017968317 0.9970414716451949 -
0.07497340408030169 -1.38388
            children [
                DEF STICK1_SHAPE Shape {
                    appearance DEF STICK_COLOR PBRAppearance {
                        baseColor 0.443137 0.431373 0.447059
                        roughness 1
                        metalness 0
                    }
                    geometry Box {
                        size 0.01 0.01 1
                    }
                }
            ]
            name "stick1"
            boundingObject USE STICK1_SHAPE
        }
        DEF STICK2 Solid {
            translation 0.327 0.154 0.148
            rotation 0.015039718542295452 0.9970551531988499
0.07519859271147726 -1.38
            children [
                DEF STICK2_SHAPE Shape {
                    appearance USE STICK_COLOR
                    geometry Box {
                        size 0.01 0.01 1
                    }
                }
            ]
            name "stick2"
            boundingObject USE STICK2_SHAPE
        }
        DEF STICK3 Solid {
            translation 0.195 -0.163 -0.02
            rotation -0.05014370195119564 0.9982887500369301
0.030086221170717384 1.23664
            children [
                DEF STICK3_SHAPE Shape {
                    appearance USE STICK_COLOR
                    geometry Box {
                        size 0.01 0.01 0.75

```

```

    }
  }
]
name "stick3"
boundingObject USE STICK3_SHAPE
}
DEF STICK4 Solid {
  translation 0.192 0.1646 -0.017
  rotation 0.06992636973228512 0.9975019914833652 -
0.010003989343473154 1.24
  children [
    DEF STICK4_SHAPE Shape {
      appearance USE STICK_COLOR
      geometry Box {
        size 0.01 0.01 0.75
      }
    }
  ]
  name "stick4"
  boundingObject USE STICK4_SHAPE
}
DEF STICK5 Solid {
  translation 0.814 5.68543e-06 0.0513
  rotation 1 0 0 1.5708
  children [
    DEF STICK5_SHAPE Shape {
      appearance USE STICK_COLOR
      geometry Box {
        size 0.01 0.01 0.23
      }
    }
  ]
  name "stick5"
  boundingObject USE STICK5_SHAPE
}
DEF STICK6 Solid {
  translation 0.663 5.50178e-06 0.083
  rotation 1 0 0 1.5708
  children [
    DEF STICK6_SHAPE Shape {
      appearance USE STICK_COLOR
      geometry Box {
        size 0.01 0.01 0.25
      }
    }
  ]
  name "stick6"
  boundingObject USE STICK6_SHAPE
}
DEF STICK7 Solid {
  translation 0.05 -1.73472e-18 0.203
  children [
    DEF STICK7_SHAPE Shape {
      appearance USE STICK_COLOR
      geometry Box {
        size 0.01 0.35 0.01

```

```

    }
  }
]
name "stick7"
boundingObject USE STICK7_SHAPE
}
DEF STICK8 Solid {
  translation 0.05 -0.177 0.067
  children [
    DEF STICK8_SHAPE Shape {
      appearance USE STICK_COLOR
      geometry Box {
        size 0.01 0.01 0.277
      }
    }
  ]
  name "stick8"
  boundingObject USE STICK8_SHAPE
}
DEF STICK9 Solid {
  translation 0.05 0.176 0.068
  children [
    DEF STICK9_SHAPE Shape {
      appearance USE STICK_COLOR
      geometry Box {
        size 0.01 0.01 0.275
      }
    }
  ]
  name "stick9"
  boundingObject USE STICK9_SHAPE
}
DEF STICK10 Solid {
  translation 0.05 5.93452e-06 -0.071
  rotation 1 0 0 -1.5707953071795862
  children [
    DEF STICK10_SHAPE Shape {
      appearance USE STICK_COLOR
      geometry Box {
        size 0.01 0.01 0.36
      }
    }
  ]
  name "stick10"
  boundingObject USE STICK10_SHAPE
}
DEF STICK11 Solid {
  translation 0.814 5.75683e-06 -0.0187
  rotation 1 0 0 -1.5707953071795862
  children [
    DEF STICK11_SHAPE Shape {
      appearance USE STICK_COLOR
      geometry Box {
        size 0.01 0.01 0.23
      }
    }
  ]
}

```

```

    ]
    name "stick11"
    boundingObject USE STICK11_SHAPE
}
DEF STICK12 Solid {
    translation 0.814 -0.1124 0.018
    rotation 1 0 0 0
    children [
        DEF STICK12_SHAPE Shape {
            appearance USE STICK_COLOR
            geometry Box {
                size 0.01 0.01 0.07
            }
        }
    ]
    name "stick12"
    boundingObject USE STICK12_SHAPE
}
DEF STICK13 Solid {
    translation 0.814 0.1124 0.018
    rotation 1 0 0 -3.141592653589793
    children [
        DEF STICK13_SHAPE Shape {
            appearance USE STICK_COLOR
            geometry Box {
                size 0.01 0.01 0.07
            }
        }
    ]
    name "stick13"
    boundingObject USE STICK13_SHAPE
}
]
name "body"
}
DEF LEFT_WHEEL HingeJoint {
    jointParameters HingeJointParameters {
        anchor 0.19 0.0200158 0.120003
    }
    device [
        RotationalMotor {
            name "left_wheel_motor"
        }
        PositionSensor {
            name "left_wheel_position_sensor"
        }
    ]
}
endPoint Solid {
    translation 0.21 0.0200205000000000042 0.12000399999999998
    rotation 1 0 0 0
    children [
        DEF RUNNING_WHEEL Transform {
            rotation 0 0 1 -1.5707996938995747
            children [
                Shape {
                    appearance PBRAppearance {

```

```

        baseColor 0.196078 0.215686 0.215686
        roughness 1
        metalness 0
    }
    geometry Cylinder {
        height 0.02
        radius 0.054
        subdivision 24
    }
}
]
rotationStep 0
}
]
name "left_wheel"
boundingObject USE RUNNING_WHEEL
physics DEF WHEEL_PHYSICS Physics {
}
}
}
DEF RIGHT_WHEEL HingeJoint {
    jointParameters HingeJointParameters {
        anchor -0.19 0.0200163 0.120003
    }
    device [
        RotationalMotor {
            name "right_wheel_motor"
        }
        PositionSensor {
            name "right_wheel_position_sensor"
        }
    ]
    endPoint Solid {
        translation -0.21 0.02002130000000006 0.12000399999999976
        rotation 1 0 0
        children [
            USE RUNNING_WHEEL
        ]
        name "RIGHT_WHEEL"
        boundingObject USE RUNNING_WHEEL
        physics USE WHEEL_PHYSICS
    }
}
}
DEF BACK_WHEEL_JOINT Hinge2Joint {
    jointParameters HingeJointParameters {
        anchor 0.01 -0.02 -0.189997
    }
    endPoint Solid {
        translation 0.01 -0.02 -0.189997
        children [
            DEF BACK_WHEEL Transform {
                children [
                    Shape {
                        appearance PBRAppearance {
                            baseColor 0.196078 0.215686 0.215686
                            roughness 1

```

```

        metalness 0
    }
    geometry Sphere {
        radius 0.017
        subdivision 5
        ico FALSE
    }
}
]
}
]
name "BACK_WHEEL"
boundingObject USE BACK_WHEEL
physics Physics {
}
}
}
DEF DS_FC DistanceSensor {
    translation -5.38735e-07 0.21 0.16
    rotation 0 1 0 -1.5707996938995747
    children [
        DEF DS_SENSOR_SHAPE Shape {
            appearance PBRAppearance {
                roughness 1
                metalness 0
            }
            geometry Box {
                size 0.01 0.02 0.04
            }
        }
    ]
    name "ds_fc"
    lookupTable [
        0 0 0
        2 255 0
    ]
}
DEF DS_FL DistanceSensor {
    translation 0.099999 0.21 0.16
    rotation 0 1 0 -1.5707996938995747
    children [
        USE DS_SENSOR_SHAPE
    ]
    name "ds_fl"
    lookupTable [
        0 0 0
        2 255 0
    ]
}
DEF DS_FR DistanceSensor {
    translation -0.1 0.21 0.16
    rotation 0 1 0 -1.5707996938995747
    children [
        USE DS_SENSOR_SHAPE
    ]
    name "ds_fr"
}

```



```

        lookupTable [
            0 0 0
            2 255 0
        ]
    }
    DEF DS_LF DistanceSensor {
        translation 0.183 0.21 0.11
        children [
            USE DS_SENSOR_SHAPE
        ]
        name "ds_lf"
        lookupTable [
            0 0 0
            2 255 0
        ]
    }
    DEF DS_LB DistanceSensor {
        translation 0.183 0.21 -0.07
        children [
            USE DS_SENSOR_SHAPE
        ]
        name "ds_lb"
        lookupTable [
            0 0 0
            2 255 0
        ]
    }
    DEF DS_RF DistanceSensor {
        translation -0.181 0.21 0.11
        rotation 0 1 0 3.141592653589793
        children [
            USE DS_SENSOR_SHAPE
        ]
        name "ds_rf"
        lookupTable [
            0 0 0
            2 255 0
        ]
    }
    DEF DS_RB DistanceSensor {
        translation -0.181 0.21 -0.07
        rotation 0 1 0 -3.141592653589793
        children [
            USE DS_SENSOR_SHAPE
        ]
        name "ds_rb"
        lookupTable [
            0 0 0
            2 255 0
        ]
    }
    Receiver {
    }
    Camera {
        translation -1.46928e-07 0.995 -0.025
        rotation 0 1 0 -3.1415853071795863
    }

```

```

        children [
            Transform {
                rotation 1 0 0 1.57
                children [
                    Shape {
                        appearance PBRAppearance {
                            baseColor 0 0 0
                            roughness 1
                            metalness 0
                        }
                        geometry Box {
                            size 0.18 0.07 0.04
                        }
                    }
                ]
            }
        ]
        width 720
        height 480
    }
]
name "elc-bot"
boundingObject USE PLATFORM
physics Physics {
    mass 1
}
controller "ELCController"
}
DEF DRIVER Robot {
    children [
        Emitter {
        }
    ]
    name "Driver"
    controller "driver"
    supervisor TRUE
}

```

Приложение В Код контроллера передатчика

```

from controller import Supervisor
class Driver(Supervisor):
    timeStep = 32
    def __init__(self):
        super(Driver, self).__init__()
        self.emitter = self.getEmitter('emitter')
        self.keyboard.enable(Driver.timeStep)
        self.keyboard = self.getKeyboard()
    def run(self):
        self.displayHelp()
        previous_message = ''
        while True:
            k = self.keyboard.getKey()
            message = ''
            if k == ord('W'):
                message = 'W'
            elif k == ord('A'):
                message = 'A'

```

```

elif k == ord('S'):
    message = 'S'
elif k == ord('D'):
    message = 'D'
elif k == ord(' '):
    message = 'STOP'
elif k == ord('1'):
    message = "1"
elif k == ord('2'):
    message = "2"
elif k == ord('3'):
    message = "3"
elif k == ord('4'):
    message = "4"
elif k == ord('5'):
    message = "5"
elif k == ord('6'):
    message = "6"
elif k == ord('7'):
    message = "7"
elif k == ord('8'):
    message = "8"
elif k == ord('9'):
    message = "9"
elif k == ord('0'):
    message = "0"
elif k == ord('M'):
    message = "AUTO"
if message != '' and message != previous_message:
    previous_message = message
    print("I SAY " + message)
    self.emitter.send(message.encode('utf-8'))
if self.step(self.timeStep) == -1:
    break

```

```

controller = Driver()
controller.run()

```

Приложение С. Код контроллера робота

```

from controller import Robot

```

```

# Константы

```

```

# Максимальное значение передних датчиков

```

```

FRONT_DISTANCE_SENSOR_MAX_VALUE = 50

```

```

# Максимальное значение боковых датчиков

```

```

SIDE_DISTANCE_SENSOR_MAX_VALUE = 40

```

```

# Соответствие списка значений с датчиков по индексу

```

```

LB = 0

```

```

LF = 1

```

```

FL = 2

```

```

FC = 3

```

```

FR = 4

```

```

RF = 5

```

```

RB = 6

```

```

# Константы для обозначения сторон

```

```

DEFAULT = 0

```

```

LEFT = -1

```

```

RIGHT = 1

```

```

class ELCBot(Robot):
    time_step = 32
    max_speed = 10.0
    def __init__(self):
        super(ELCBot, self).__init__()
        self.left_motor = self.getMotor('left_wheel_motor')
        self.right_motor = self.getMotor('right_wheel_motor')
        self.left_ps = self.getPositionSensor('left_wheel_position_sensor')
        self.right_ps = self.getPositionSensor('right_wheel_position_sensor')
        self.ds_fc = self.getDistanceSensor('ds_fc')
        self.ds_fl = self.getDistanceSensor('ds_fl')
        self.ds_fr = self.getDistanceSensor('ds_fr')
        self.ds_lf = self.getDistanceSensor('ds_lf')
        self.ds_lb = self.getDistanceSensor('ds_lb')
        self.ds_rf = self.getDistanceSensor('ds_rf')
        self.ds_rb = self.getDistanceSensor('ds_rb')
        self.receiver = self.getReceiver('receiver')
        self.left_motor.setPosition(float('inf'))
        self.left_motor.setVelocity(0.0)
        self.right_motor.setPosition(float('inf'))
        self.right_motor.setVelocity(0.0)
        self.ds_fc.enable(self.time_step)
        self.ds_fl.enable(self.time_step)
        self.ds_fr.enable(self.time_step)
        self.ds_lf.enable(self.time_step)
        self.ds_lb.enable(self.time_step)
        self.ds_rf.enable(self.time_step)
        self.ds_rb.enable(self.time_step)
        self.receiver.enable(self.time_step)
        self.camera = self.getCamera('camera')
        self.camera.enable(4 * self.time_step)

class Settings:
    message = "STOP"
    prev_message = "STOP"
    speed = 0
    auto_move = False
    block_forward = False
    turn_mode = DEFAULT
    divide_deviant = 0.2
    on_wall = DEFAULT
    kd_for_wall = 0.975
    kd_for_outer_turn = 2

class PID:
    kp = 0.1
    ki = 0.022
    kd = 0.97
    integral = 0.0
    old_y = 0.0
    integral_min = -1.0
    integral_max = 1.0

elc_bot = ELCBot()
settings = Settings()
pid = PID()
def stop():
    global elc_bot
    elc_bot.left_motor.setVelocity(0)

```

```

    elc_bot.right_motor.setVelocity(0)
def move_forward():
    global elc_bot
    elc_bot.left_motor.setVelocity(settings.speed)
    elc_bot.right_motor.setVelocity(settings.speed)
def move_backward():
    global elc_bot
    elc_bot.left_motor.setVelocity(-settings.speed)
    elc_bot.right_motor.setVelocity(-settings.speed)
def turn_left():
    global elc_bot
    elc_bot.left_motor.setVelocity(-settings.speed)
    elc_bot.right_motor.setVelocity(settings.speed)
def turn_right():
    global elc_bot
    elc_bot.left_motor.setVelocity(settings.speed)
    elc_bot.right_motor.setVelocity(-settings.speed)
def speed_handler():
    global elc_bot, settings
    def set_speed(new_speed):
        print('MY SPEED = ', new_speed, '!')
        settings.speed = new_speed
        settings.message = settings.prev_message
    if settings.message == "1":
        set_speed(1)
    elif settings.message == "2":
        set_speed(2)
    elif settings.message == "3":
        set_speed(3)
    elif settings.message == "4":
        set_speed(4)
    elif settings.message == "5":
        set_speed(5)
    elif settings.message == "6":
        set_speed(6)
    elif settings.message == "7":
        set_speed(7)
    elif settings.message == "8":
        set_speed(8)
    elif settings.message == "9":
        set_speed(9)
    elif settings.message == "0":
        set_speed(10)
def move_handler():
    global settings
    if settings.message == 'W':
        print('I MOVE FORWARD!')
        if front_obstacle():
            stop()
            return
        move_forward()
    elif settings.message == 'A':
        print('I TURN LEFT!')
        if left_obstacle():
            stop()
            return

```

```

        turn_left()
    elif settings.message == 'S':
        print('I MOVE BACKWARD!')
        move_backward()
    elif settings.message == 'D':
        print('I TURN RIGHT!')
        if right_obstacle():
            stop()
            right_equal()
        turn_right()
    elif settings.message == 'STOP':
        print('I STOP!')
        stop()
def front_obstacle():
    global ds_values
    return ds_values[FL] < FRONT_DISTANCE_SENSOR_MAX_VALUE or \
           ds_values[FC] < FRONT_DISTANCE_SENSOR_MAX_VALUE or \
           ds_values[FR] < FRONT_DISTANCE_SENSOR_MAX_VALUE
def left_obstacle():
    global ds_values
    return ds_values[LF] < SIDE_DISTANCE_SENSOR_MAX_VALUE or \
           ds_values[LB] < SIDE_DISTANCE_SENSOR_MAX_VALUE
def right_obstacle():
    global ds_values
    return ds_values[RF] < SIDE_DISTANCE_SENSOR_MAX_VALUE or \
           ds_values[RB] < SIDE_DISTANCE_SENSOR_MAX_VALUE
def left_equal():
    global ds_values
    return abs(ds_values[LF] - ds_values[LB]) <= settings.divide_deviant
def right_equal():
    global ds_values
    return abs(ds_values[RF] - ds_values[RB]) <= settings.divide_deviant
def end_wall():
    global ds_values
    if ds_values[RF] > SIDE_DISTANCE_SENSOR_MAX_VALUE + 20 >= ds_values[RB]:
        return RIGHT
    if ds_values[LF] > SIDE_DISTANCE_SENSOR_MAX_VALUE >= ds_values[LB]:
        return LEFT
    return DEFAULT
def evaluate_pid(distance):
    global pid
    error = distance - SIDE_DISTANCE_SENSOR_MAX_VALUE
    up = pid.kp * error
    pid.integral -= error
    if pid.integral > pid.integral__max:
        pid.integral = pid.integral__max
    else:
        if pid.integral < pid.integral_min:
            pid.integral = pid.integral_min
    ui = pid.ki * pid.integral
    ud = pid.kd * error - pid.old_y
    us = up + ui + ud
    pid.old_y = error
    if us >= 10:
        us = 10
    if us <= -10:

```

```

        us = -10
    return us
while elc_bot.step(elc_bot.time_step) != -1:
    ds_values = [elc_bot.ds_lb.getValue(),
                  elc_bot.ds_lf.getValue(),
                  elc_bot.ds_fl.getValue(),
                  elc_bot.ds_fc.getValue(),
                  elc_bot.ds_fr.getValue(),
                  elc_bot.ds_rf.getValue(),
                  elc_bot.ds_rb.getValue()]
    if elc_bot.receiver.getQueueLength() > 0:
        settings.prev_message = settings.message
        settings.message = elc_bot.receiver.getData().decode('utf-8')
        elc_bot.receiver.nextPacket()
        settings.auto_move = False
        speed_handler()
        move_handler()
        if settings.message == "AUTO":
            print("AUTO MOVE MODE!")
            settings.auto_move = True
    if settings.auto_move:
        if not settings.block_forward:
            if front_obstacle():
                settings.block_forward = True
                continue
        if settings.on_wall == DEFAULT:
            if not front_obstacle():
                if right_obstacle():
                    pid.kd = settings.kd_for_wall
                    settings.on_wall = RIGHT
                    continue
                if left_obstacle():
                    pid.kd = settings.kd_for_wall
                    settings.on_wall = LEFT
                    continue
            settings.speed = 10
            move_forward()
            continue
        if settings.on_wall == RIGHT:
            if end_wall() == RIGHT:
                pid.kd = settings.kd_for_outer_turn
            if end_wall() == DEFAULT:
                pid.kd = settings.kd_for_wall
            settings.speed = 10
            u = evaluate_pid(ds_values[RF])
            if u < 0:
                elc_bot.left_motor.setVelocity(settings.speed - abs(u))
                elc_bot.right_motor.setVelocity(settings.speed)
            if u > 0:
                elc_bot.left_motor.setVelocity(settings.speed)
                elc_bot.right_motor.setVelocity(settings.speed - abs(u))
            if u == 0:
                elc_bot.left_motor.setVelocity(settings.speed)
                elc_bot.right_motor.setVelocity(settings.speed)
        if settings.on_wall == LEFT:
            settings.speed = 10

```

```

    u = evaluate_pid(ds_values[LF])
    if u < 0:
        elc_bot.left_motor.setVelocity(settings.speed)
        elc_bot.right_motor.setVelocity(settings.speed - abs(u))
    if u > 0:
        elc_bot.left_motor.setVelocity(settings.speed - abs(u))
        elc_bot.right_motor.setVelocity(settings.speed)
    if u == 0:
        elc_bot.left_motor.setVelocity(settings.speed)
        elc_bot.right_motor.setVelocity(settings.speed)
    if settings.block_forward:
        if settings.turn_mode == DEFAULT:
            if not left_obstacle():
                settings.turn_mode = LEFT
                settings.speed = 2
                turn_left()
                continue
            if not right_obstacle():
                settings.turn_mode = RIGHT
                settings.speed = 2
                turn_right()
                continue
        if right_equal() and settings.turn_mode == LEFT and
right_obstacle():
            settings.turn_mode = DEFAULT
            settings.block_forward = False
            settings.on_wall = RIGHT
            stop()
            continue
        if left_equal() and settings.turn_mode == RIGHT and
left_obstacle():
            settings.turn_mode = DEFAULT
            settings.block_forward = False
            settings.on_wall = LEFT
            stop()
            continue
    pass

```