Санкт–Петербургский государственный университет

Системное программирование

Ершов Владислав Евгеньевич

Симуляция робота с учетом особенностей реальных условий работы датчиков и агрегатов

Отчет по учебной практике

Научный руководитель:
д. ф.-м. н., профессор кафедры системного программирования СПбГУ
Граничин Олег Николаевич

Консультант:
к. ф.-м. н., директор НОЦ СПбГУ “Математическая робототехника и искусственный интеллект” Амелин Константин Сергеевич

Санкт-Петербург
2021 г.

Содержание

1. Введение

Мультиагентная парадигма – один из вариантов построения вычислительных систем. Взаимодействие автономных объектов (агентов) является главной отличительной чертой такого построения. Мультиагентная парадигма предоставляет ряд преимуществ, которые позволяют успешно выполнять различные задачи за счет уменьшения размера пакетов информации, подлежащих передаче, которые требуются для работоспособности систем, между автономными объектами. Хорошим примером такой системы является Uber.

Реализация такого алгоритма авторства О. Н. Граничина и является основной задачей проекта «Рой роботов».

Алгоритм состоит в следующем: есть рой роботов, где каждый робот – агент, задача роботов добраться до определенной цели обходя препятствия. Первый робот идет по определенной траектории, передает информацию о своем курсе и уверенности в правильности курса, следующие за ним роботы, используя эти данные, улучшает свою траекторию.

До начала моей работы была симуляция в Webots и реальный робот.

Они имеют много различий, что не позволяет отработанный на симуляции алгоритм использовать в реальном роботе.

В симуляции многие физические процессы идеализированы, например, у реальных датчиков и агрегатов существуют погрешности в работе и изменение показаний, вызванные различными шумами, что приводит к необходимости уже отработанный в симуляции алгоритм кардинально менять для реального робота. Заниматься таким изменением на реальном роботе очень неэффективно, поэтому появилась необходимость в моем проекте.

Основной задачей работы является именно воплощение в симуляторе вышеописанных физических особенностей, а также доработка алгоритма для такой симуляции.

2. Постановка задачи

Целью данной работы является создание виртуального прототипа робота в симуляторе Webots с приближенными к реальным особенностям работы датчиков и агрегатов, исследование поведения такого прототипа и изменение алгоритма для такой симуляции с учетом изученных особенностей.

Поставленная цель обусловила решение следующих задач:

1. Обзор технических материалов по конструкции и особенностям робота и датчиков [2]
2. Обзор симулятора Webots [3]
3. Реализация робота
4. Проведение замеров на реальных агрегатах
5. Реализация датчиков и агрегатов, максимально приближенных к реальным
6. Приспособление алгоритма для реализуемой симуляции
7. Проверка работоспособности симуляции

3. Обзор симулятора Webots

Информация для обзора взята из статьи [1].

Кроссплатформенная система моделирования Webots является симулятором роботов, имеющим множество возможных интеграций: ROS, MATLAB и API для языков программирования C++, Java, Python.

Webots предоставляет пользователю обширную библиотеку моделей сенсоров и агрегатов, интерактивную 3D-визуализацию, набор готовых роботов и окружений, инструменты для коммуникаций между роботами.

Среди имеющихся датчиков и агрегатов есть наиболее используемые сенсоры и исполнительные устройства: датчики света, датчики определения расстояния, акселерометры, камеры и некоторые другие.

С помощью Webots можно реализовывать модель с параметрами, позволяющими протестировать ее в симуляции среды с учетом физических эффектов и включающими в себя графические свойства, такие как форма, размер, положение и ориентация, цвет и текстура объекта.

4. Реализация

4.1 Схема робота

Рис. 1: Модель реального робота

Рис. 2: Датчики и агрегаты робота

Модели агрегатов робота:

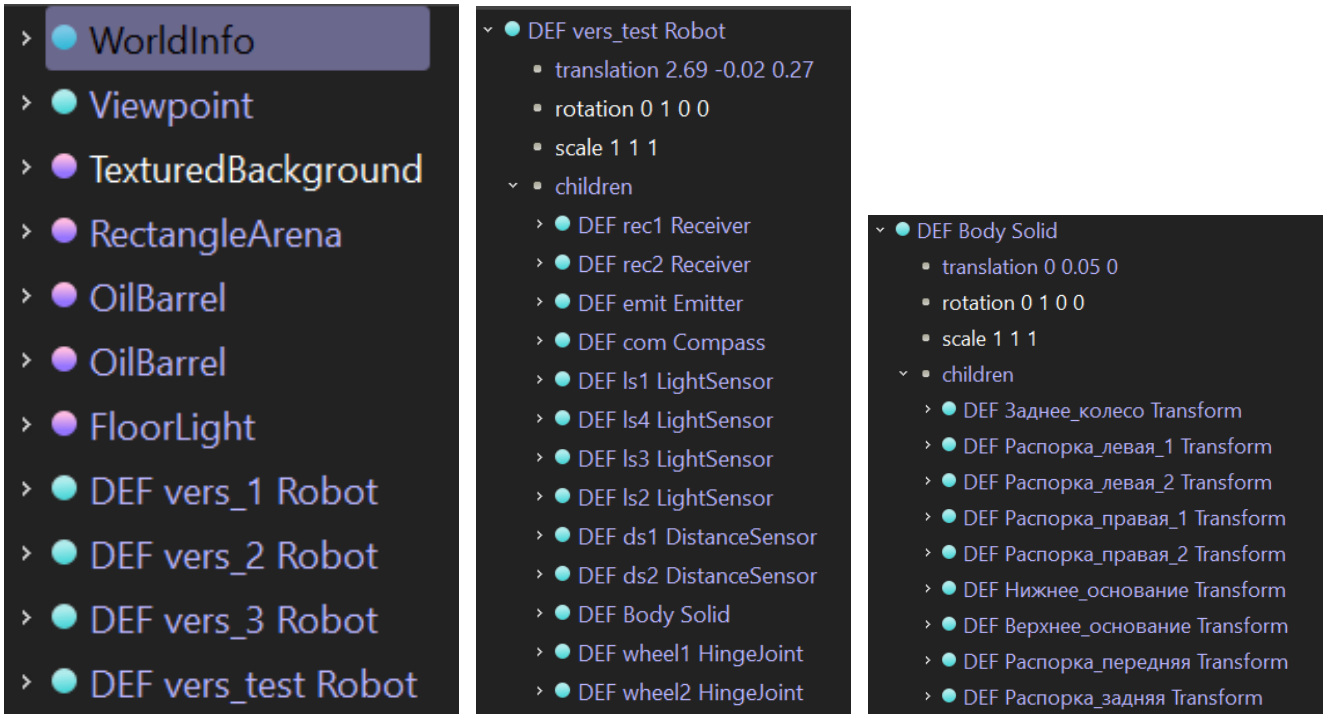
1. Магнитометр – модель HMC5883L
2. Фоторезистор – модель GL5506
3. Микрокомпьютер – Arduino Nano
4. Модуль передачи данных – модель ESP8266
5. Инфракрасный дальномер – модель Sharp GP2Y0A02YK0F

Микрокомпьютер Arduino Nano рекомендован к работе со всеми вышеперечисленными моделями датчиков.

4.2 Реализация модели в симуляции

Создание симуляции с нуля, требует добавления различных объектов: заднего плана, пола, окружения, света и т.д. (Рис. 3a).

Модель робота – это иерархическая структура, состоящая из встроенных в Webots классов с готовой функциональностью, является объектом класса robot (Рис. 3b).



(a) (b) (c)

Рис. 3

Используются классы:

1. Solid – твердая фигура сложной формы с возможностью перемещения и вращения относительно родительского объекта, есть поля, задающие физику. Пример такого объекта – тело робота (Рис. 3c)

2. Transform – надстройка над объектом, позволяющая перемещать и вращать его, пример – распорка задняя (Рис. 4a)

3. Shape – внешний вид, задает форму и внешность (Рис. 4a)

4. HingeJoint – подвижное соединение двух объектов с возможностью подключения движителя, включает в себя:

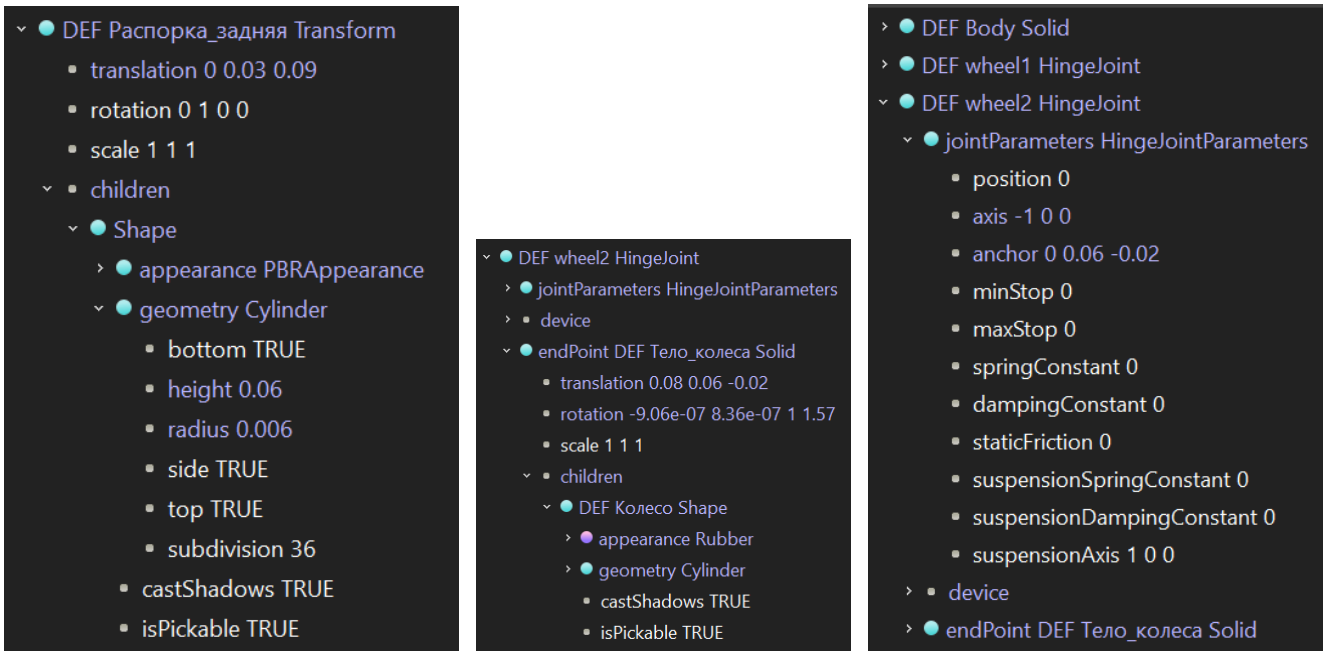
(a) endPoint – позволяет задать тело двигающегося объекта (Рис. 4b)

(b) jointParameters – задает ось вращения и точку вращения (Рис. 4c)

(c) device – агрегат, которым будет двигаться объект (Рис. 5a)

Пример – колесо и робот

5. Также есть различные датчики, которые являются улучшенным Transform, они дополнительно имеют различные поля для задания параметров своей работы

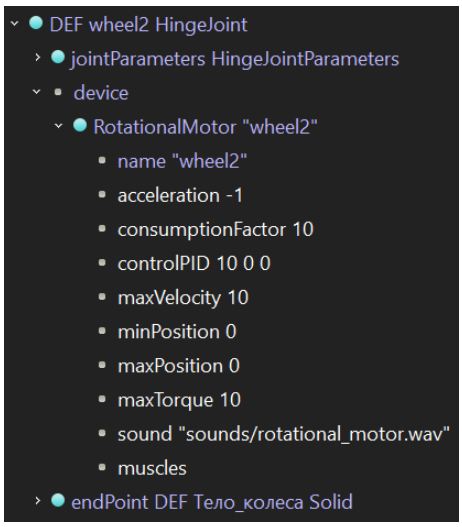


(a) **(b)** **(c)**

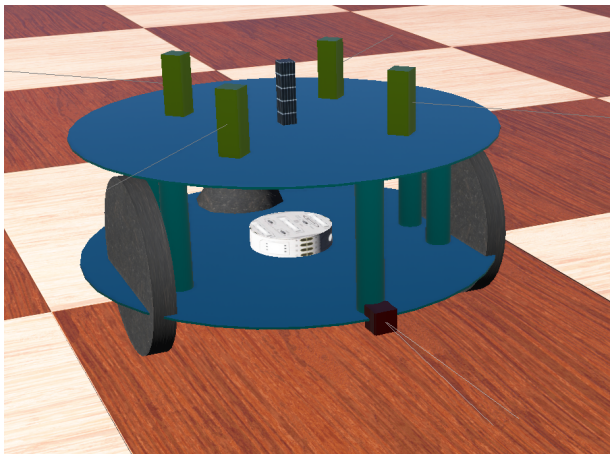
Рис. 4

Модель имеет размеры тележки 10*10*8 см и агрегаты:

- Два колеса-мотора и одно заднее пассивное колесо
- Магнитометр
- Четыре фоторезистора



(a)



(b) Модель реализованная в Webots

Рис. 5

- Модуль приемо-передачи данных
- Инфракрасный дальномер

4.3 Реализация датчиков в симуляции

4.3.1 Инфракрасный дальномер

Инфракрасный дальномер Sharp имеет угол обзора в 6 градусов. В симуляторе задаем соответствующие параметры: количество лучей и угол апертуры датчика, таблица измерений. Тип датчика оставляем по умолчанию, так как реальный датчик почти не чувствителен к изменению цветов объектов, а инфракрасный датчик в Webots чувствителен (Рис. 6).

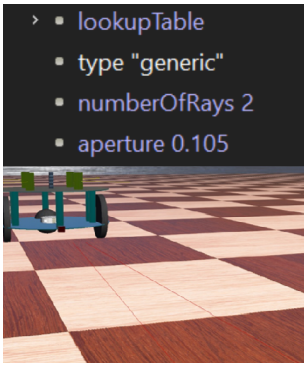


Рис. 6

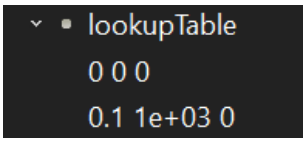


Рис. 7

Для настройки датчика заполняется таблица измерений, которую предоставляет класс DistanceSensor (Рис. 7).

- Первый столбец говорит какое расстояние должен замерять датчик в метрах
- Второй задает результат, возвращающийся из датчика, в зависимости от измеренного расстояния
- Третий задает погрешность измерения с нормальным распределением

По двум первым столбцам строится непрерывное отображение.

По умолчанию в данной таблице идет преобразование измеренного расстояния в метрах в условные единицы, в соотношении 1 к 10000.

Для данного датчика проведены замеры и определена погрешность на реальном датчике. Сделано сравнение с помехами представляемыми средствами Webots (Рис. 8).

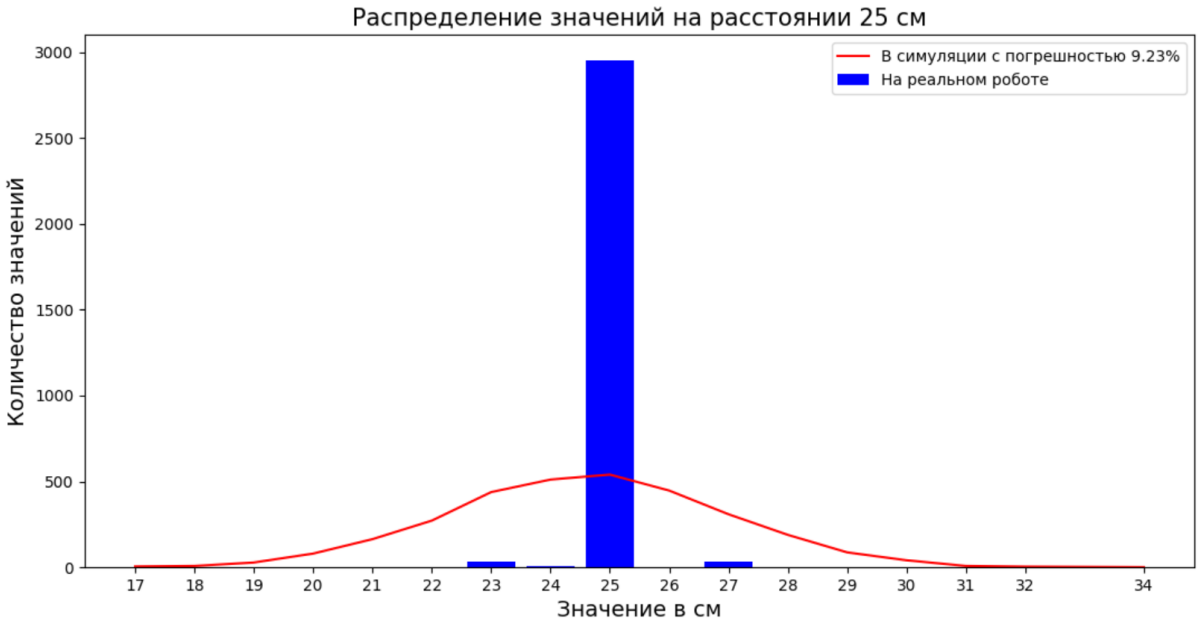
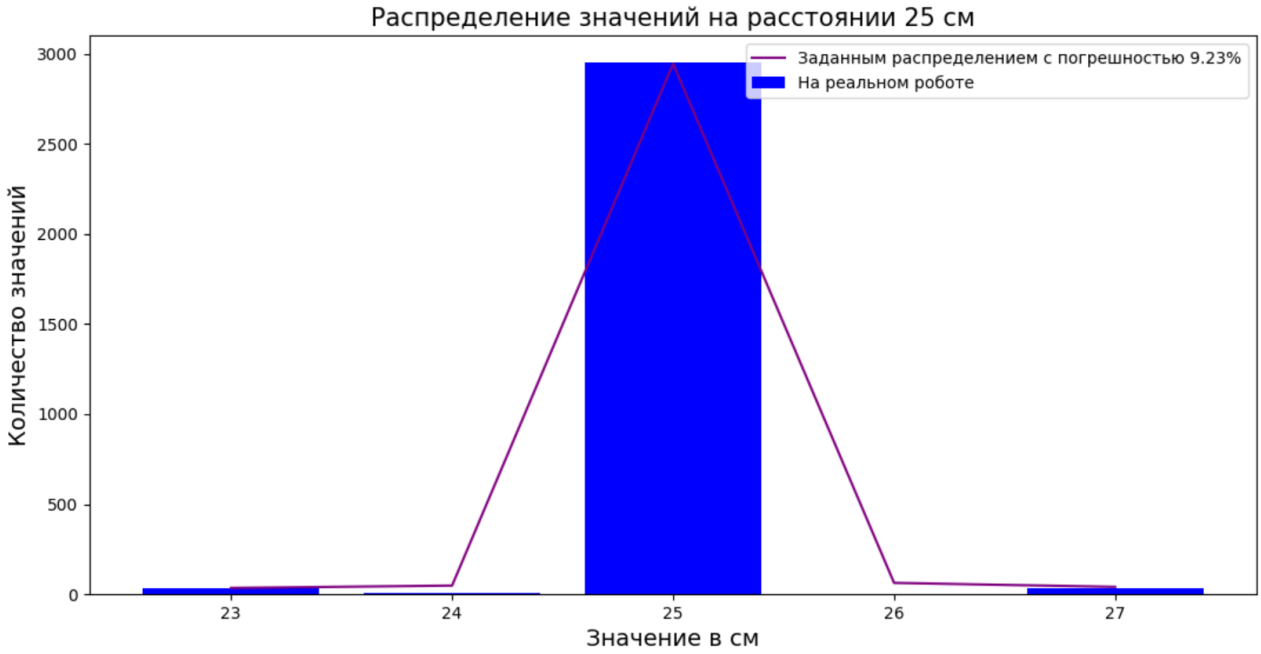


Рис. 8

Для исправления такого расхождения реализована функция (Рис. 9a), приведен результат ее работы (Рис. 9b).

```python
# Внесение погрешности в показания ds
def inaccuracy_ds(d, error_comp=0.0923):
    p = random.uniform(0, 1)
    if p <= 0.03827:
        q = random.uniform(0, error_comp)
        return round(d * (1 - q))
    elif p >= 0.96173:
        q = random.uniform(0, error_comp)
        return round(d * (1 + q))
    else:
        return round(d)
```

(a)



(b)

Рис. 9

В Webots уже есть реализация дальномеров Sharp, однако в ней не учитываются угол обзора, также там неверно задается распределение погрешностей.

В перспективе есть возможность реализовать такой датчик одним блоком. Для этого проведено изучение характеристик датчика и создан

шаблон для дальнейшего улучшения.

Реальный датчик выдает вольтаж согласно приведенной в документации [4] таблицы (Рис. 10), реализован идентичный с реальным датчиком порядок вывода данных (Рис. 11).

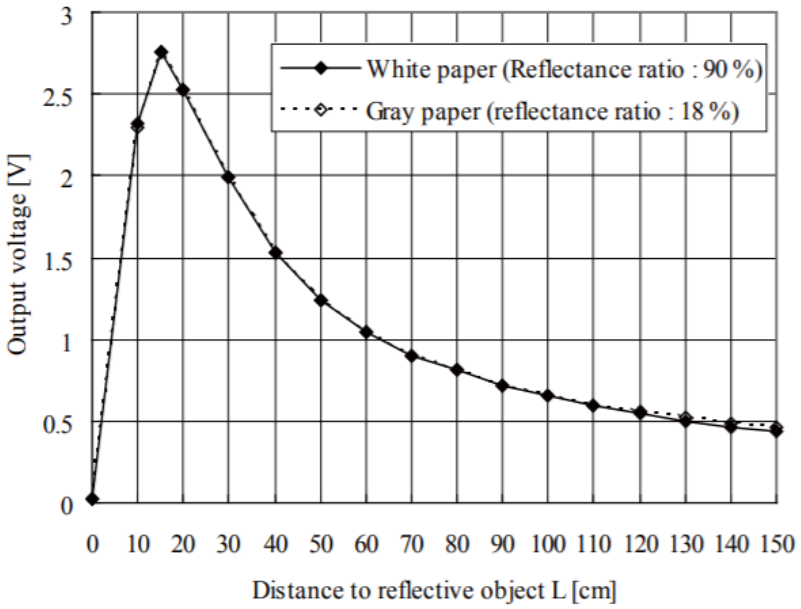


Рис. 10

```
lookupTable
0.15 2.7 0
0.2 2.48 0
0.3 1.99 0
0.4 1.54 0
0.5 1.27 0
0.6 1.04 0
0.7 0.87 0
0.8 0.8 0
0.9 0.74 0
1 0.69 0
1.1 0.61 0
1.2 0.55 0
1.3 0.5 0
1.4 0.47 0
1.5 0.45 0
```

Рис. 11

4.3.2 Фоторезистор

Расчет значений интенсивности свечения в симуляции и на реальном роботе в зависимости от угла поворота датчика относительно направления на источник света отличаются.

Проведены замеры на реальном роботе и в симуляции (Рис. 12).

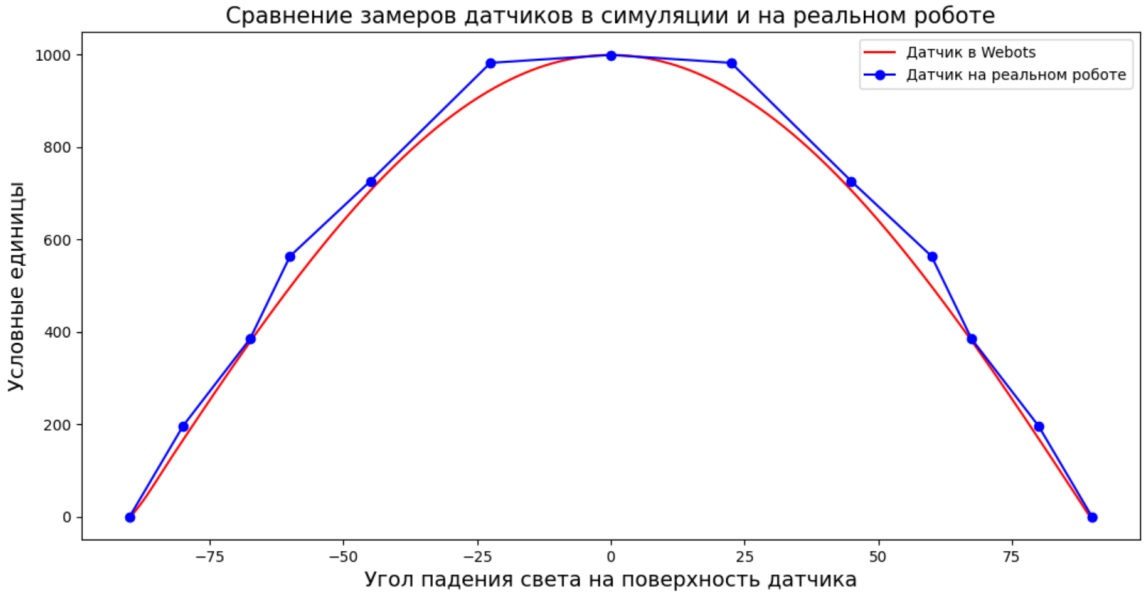


Рис. 12

Создан алгоритм, получающий на входе данные из датчика в симуляторе, который преобразует данные в соответствии с реальным датчиком (Рис. 13).

```
# Преобразование данных ls в соответствии с реальным датчиком
def transform_light(light):
    if light > 1000: light = 1000
    angle = 180 * (math.pi / 2 - math.asin(light / 1000)) / math.pi
    light_val = [0, 23, 45, 66, 85, 115, 117, 115,
                 85, 66, 45, 23, 0]
    angle_val = [-90, -80, -67.5, -60, -45, -22.5, 0,
                 22.5, 45, 60, 67.5, 80, 90]
    return numpy.interp(angle, angle_val, light_val) * 1000 / 117
```

Рис. 13

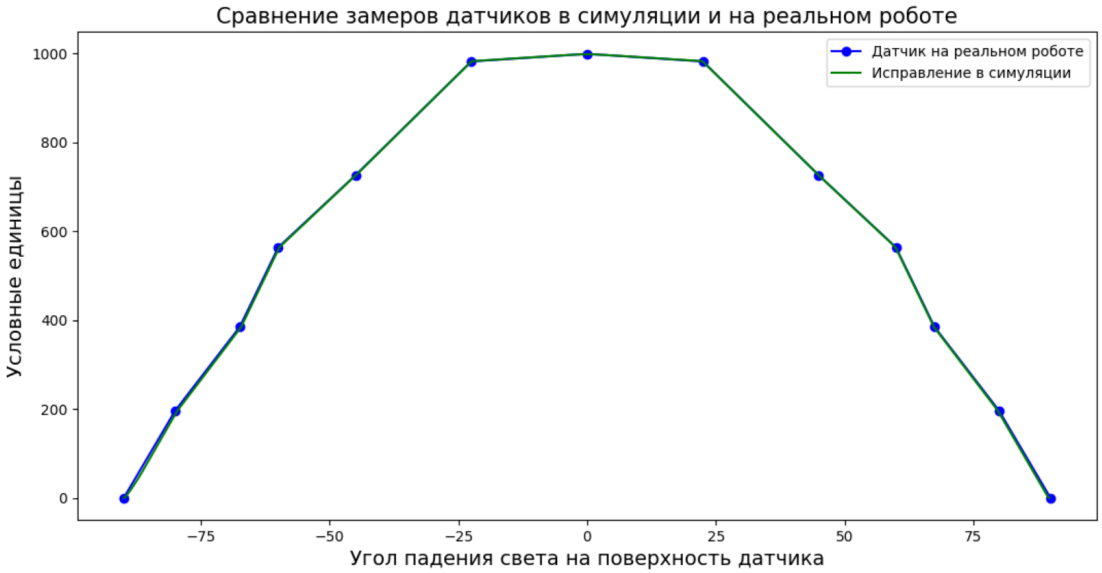


Рис. 14: Результат работы вышеприведенного алгоритма

Для фоторезистора проведены замеры и определена погрешность на реальном датчике. Сделано сравнение с помехами представляемыми средствами Webots (Рис. 15).

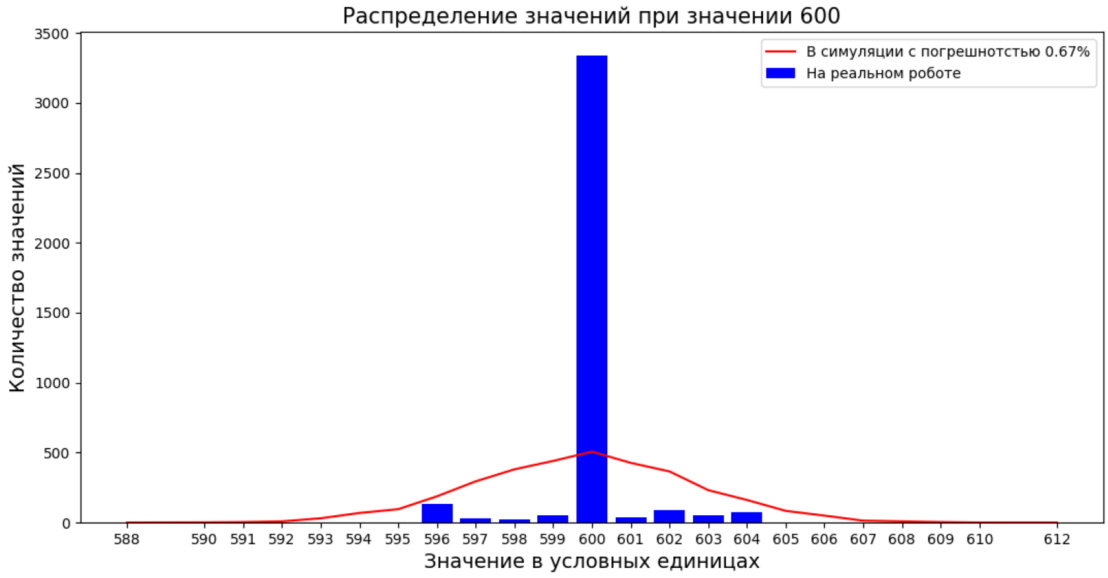


Рис. 15

Для исправления такого расхождения реализована функция (Рис. 16).

```python
# Внесение погрешности в показания ls
def inaccuracy_ls(light):
    percent_val = [0.7951, 0.7985, 0.8152, 0.8489, 0.8723, 0.9046]
    error_comp_val = [0.4706, 0.198, 0.0796, 0.0175, 0.0067, 0.0059]
    light_val = [51, 101, 201, 399, 600, 799]

    error_comp = numpy.interp(light, light_val, error_comp_val)
    percent = numpy.interp(light, light_val, percent_val)

    p = random.uniform(0, 1)
    if p <= (1 - percent) / 2:
        q = random.uniform(0, error_comp)
        return round(light * (1 - q))
    elif p >= percent / 2 + 0.5:
        q = random.uniform(0, error_comp)
        return round(light * (1 + q))
    else:
        return round(light)
```

Рис. 16

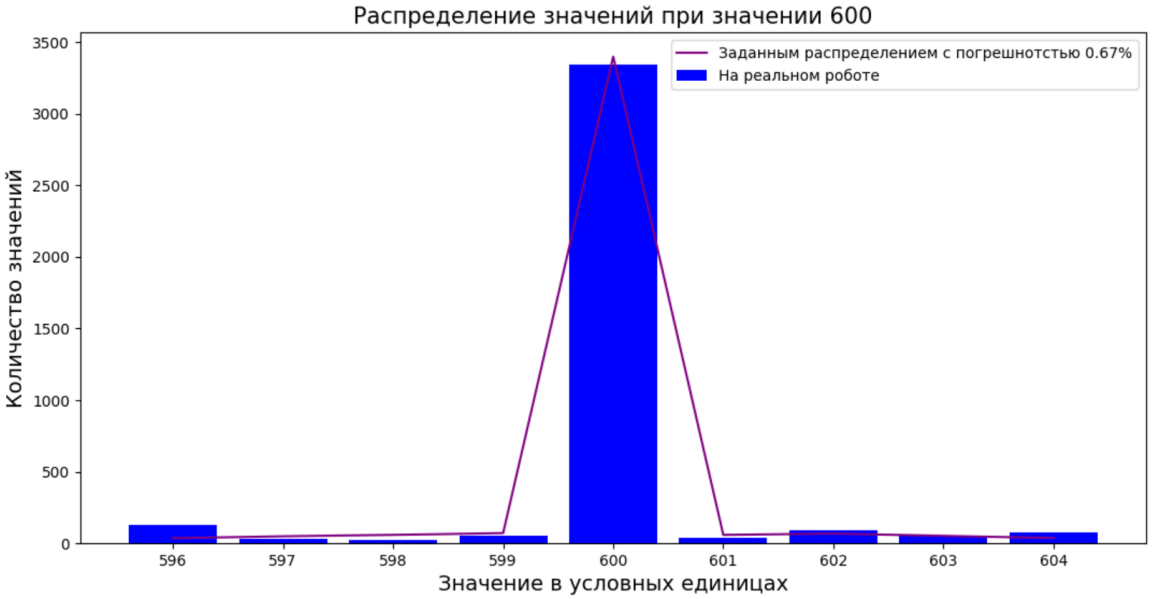


Рис. 17: Результат его работы

4.3.3 Магнитометр

Коэффициент declination, вносящийся в магнитометры для более точного определения направления на магнитный полюс в конкретном месте, в данной симуляции не нужен, так как интересно только определение направления относительно других роботов. Все роботы находятся в компактной группе в одном месте, следовательно внесение данной константы не требуется.

На реальном роботе провести замеры магнитометра не представляется возможным, так как его использование в помещениях вносит слишком большие искажения в работе, а вне помещения нет смысла, так как работе алгоритма мешает большое количество фонового света.

4.3.4 Приемо-передающее устройство

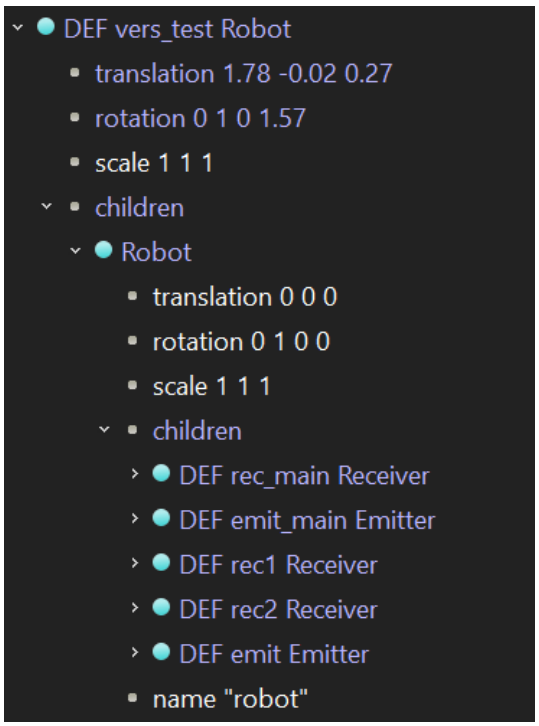
В реальном роботе есть два микроконтроллера – один отвечает за передачу сообщений другим роботам, второй за все остальные действия. Их работа асинхронна. Они обмениваются информацией с помощью проводной связи, задержки и помехи отсутствуют. В симуляции же была реализована только одна управляющая программа, которая управляла всеми процессами в роботе.

Проделана следующая работа:

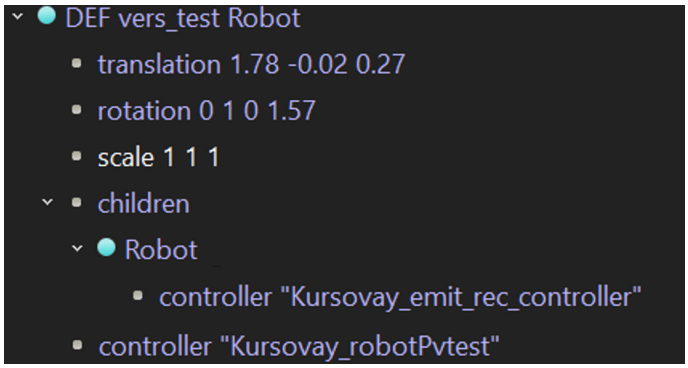
1. Создана новая программа, отвечающая за связь роботов, и перенесены все ее функции из старого контроллера

2. Внесен новый блок в робота, отвечающий за связь (Рис. 18a) и к нему подключена программа (Рис. 18b)

3. Создан алгоритм взаимодействия двух программ, симулирующий взаимодействие между реальными микроконтроллерами (Рис. 19)



(a)



(b)

Рис. 18

```
# Принимаем сообщение от основного микрочипа
if rec_main.getQueueLength() > 0:
    message = rec_main.getData()
    dataList = struct.unpack("dd", message)
    bearing = dataList[0]
    q = dataList[1]
    rec_main.nextPacket()
# Перадаем сообщение на основной микрочип
for i in range(num_of_robots):
    message = struct.pack("dd", bearingn[i][0], bearingn[i][1])
    emit_main.send(message)
```

Рис. 19

На реальных роботах передача данных осуществляется при помощи радио. Пока нет конечного выбора протокола передачи сообщений. Работа по проведению замеров погрешностей и задержек остается на перспективу. Для реализации особенностей используются справочные данные протокола UDP [7].

Симуляция разной скорости работы микроконтроллеров осуществляется с помощью установки разного timestep в двух программах.

Приемо-передающее устройство использует протокол UDP. В нем

есть потери сообщений по разным причинам, но реагирование робота на потери одинаково, то есть нам неважно из-за чего сообщение не дошло. В симуляции реализована потеря сообщений с определенной вероятностью (Рис 20).

```
# Передаем сообщение соседям
p = random.uniform(0, 1)
if p > 0.04:
    message = struct.pack("dd", bearing, q)
    emit.send(message)
```

Рис. 20

Также приемо-передающее устройство затрачивает определенное время на переключение между соседними роботами для получения от них сообщения. Реализована задержка при таком переключении (Рис. 21).

```
# Принимаем сообщение от соседей
if time_wait_switch != 0:
    time_wait_switch -= 1
elif rec[index_robot_rec].getQueueLength() > 0:
    message = rec[index_robot_rec].getData()
    dataList = struct.unpack("dd", message)
    bearingn[index_robot_rec][0] = dataList[0]
    bearingn[index_robot_rec][1] = dataList[1]
    rec[index_robot_rec].nextPacket()

    index_robot_rec += 1
    index_robot_rec %= num_of_robots
    time_wait_switch = 2
```

Рис. 21

4.4 Реализация алгоритма

4.4.1 Определение расстояния до препятствия

На перспективу реализована функция, преобразующая сигнал дальномера в расстояние с погрешностью 2 сантиметра (Рис. 22), приведен результат ее работы (Рис. 23).

```
# Преобразование данных дистанционного сенсора в расстояние
def transform_d(volt):
    volt_val = [0.45, 0.47, 0.5, 0.55, 0.61, 0.69, 0.74,
                0.8, 0.87, 1.04, 1.27, 1.54, 1.99, 2.48, 2.7]
    dist_val = [150, 140, 130, 120, 110, 100, 90,
                80, 70, 60, 50, 40, 30, 20, 15]
    return numpy.interp(volt, volt_val, dist_val)
```

Рис. 22

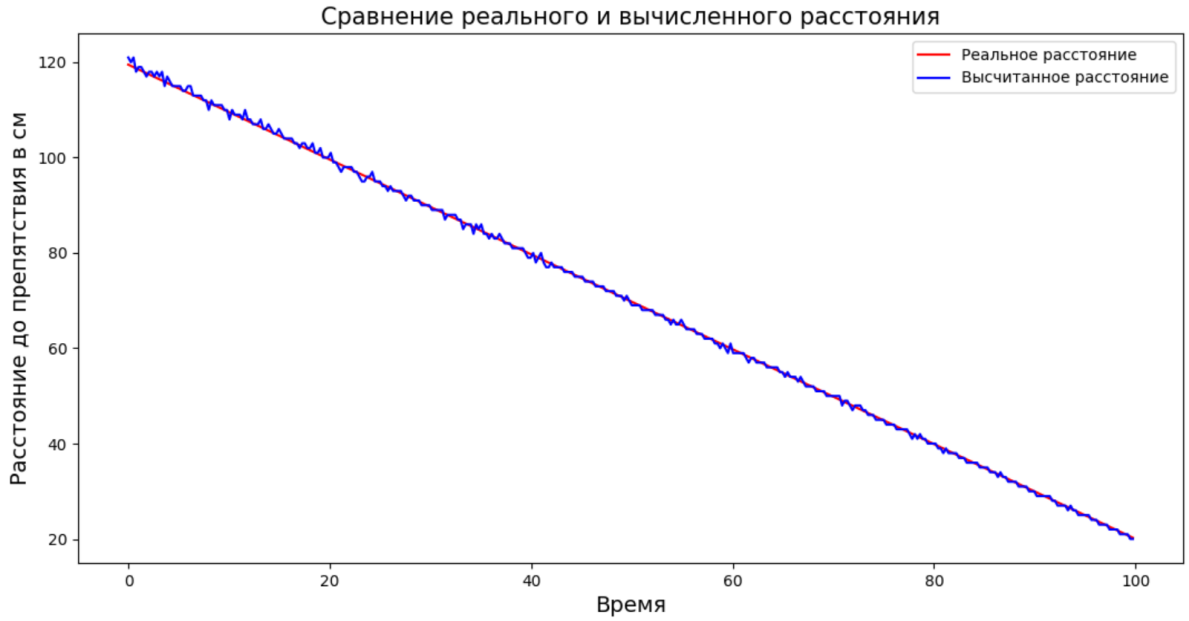


Рис. 23

4.4.2 Определение направления на источник света

Для определения направления на источник света требуется создать алгоритм, находящий источник света по фоторезисторам.

Реализована функция, вычисляющая угол с погрешностью в 2 градуса (Рис. 24), приведен ее результат (Рис. 25).

```python
# Нахождение угла по датчикам ls
def comp_angle(light):
    if max(light) == sum(light):
        return light.index(max(light)) * 90 + 45

    if light[0] + light[1] == 0:
        real_l1 = light[2]
        real_l2 = light[3]
        angle_sector = 225
    if light[1] + light[2] == 0:
        real_l1 = light[3]
        real_l2 = light[0]
        angle_sector = 315
    if light[2] + light[3] == 0:
        real_l1 = light[0]
        real_l2 = light[1]
        angle_sector = 45
    if light[3] + light[0] == 0:
        real_l1 = light[1]
        real_l2 = light[2]
        angle_sector = 135

    return (math.atan(real_l2 / real_l1) * 180 / math.pi + angle_sector) % 360
```

Рис. 24

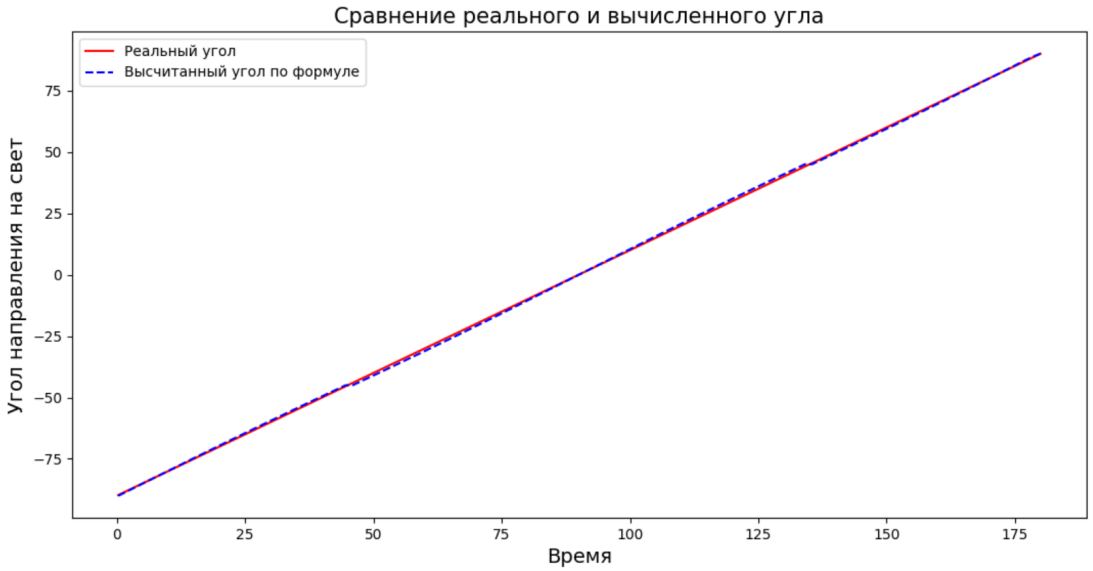


Рис. 25

5. Заключение

Были достигнуты следующие результаты:

1. Обзор технических материалов по конструкции и особенностям робота [2]
2. Обзор симулятора Webots [3]
3. Реализация робота
4. Проведение замеров на реальных агрегатах
5. Реализация датчиков и агрегатов, максимально приближенных к реальным
6. Приспособление алгоритма для реализуемой симуляции
7. Проверка работоспособности симуляции

Код проекта доступен на сайте Github:
https://github.com/ErshovVladislav10M/Roi_robotov

Список литературы

[1] M.A. Rowbo E.E. Ovsyannikova A.A. Chumachenko. Review of simulation tools for teams of robots with elements of social organization. – 2017. – (дата обращения 2021-05-24).
http://simulatdefaultion.su/uploads/files/default/2017-rovbo-ovsyannikova-chumachenko.pdf

[2] К.С. Амелин Н.О. Амелина. Проект "Рой из 100 роботов". – 2020. – (дата обращения 2021-05-24).
https://www.elibrary.ru/item.asp?id=42834567

[3] Webots. Documentation. Guide. – 2020 – (дата обращения 2021-05-24).
https://cyberbotics.com/doc/guide/index

[4] Sharp GP2Y0A02YK0F. Documentation. – 2006 – (дата обращения 2021-05-24).
http://robocraft.ru/files/sensors/Sharp/GP2Y0A02YK0F/SHARP-GP2Y0A02YK0F.pdf

[5] GL55 Series. Documentation. – 2008 – (дата обращения 2021-05-24).
https://pdf.voron.ua/files/pdf/nysenba/GL55%20Series.pdf

[6] ESP8266. Documentation. – 2016 – (дата обращения 2021-05-24).
https://www.euromobile.ru/upload/iblock/38e/38edea9ed541014c941ac8a47619db6

[7] Протокол UDP. – 2021 – (дата обращения 2021-05-24).
https://ru.wikipedia.org/wiki/UDP