Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет «Информационных систем и технологий»

Кафедра «Вычислительная техника»

Дисциплина «Программная инженерия»

**КУРСОВОЙ ПРОЕКТ (РАБОТА)**

Тема «Разработка программно доступных компонент для организации робототехнических соревнований в виртуальной среде» »

Выполнил студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /П.С. Кондратьев /

подпись инициалы, фамилия

Курс 1 Группа ИВТАСмд-11

Направление/ специальность 09.04.01 «Информатика и вычислительная техника»

Руководитель доцент Святов К.В.

Дата сдачи:

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

Дата защиты:

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ульяновск

2021 г.

Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет «Информационных систем и технологий»

Кафедра «Вычислительная техника»

Дисциплина «Программная инженерия»

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)**

студенту ИВТАСмд-11 Кондратьев П.С.

группа фамилия, инициалы

Тема проекта (работы) «Разработка программно доступных компонент для организации робототехнических соревнований в виртуальной среде»

Срок сдачи законченного проекта (работы) «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

Исходные данные к проекту (работе) Webots .

Содержание пояснительной записки (перечень подлежащих разработке вопросов)

Пояснительная записка должна содержать 25 – более страниц текста, распределенного по разделам следующим образом:

1. Титульный лист

2. Введение

3. Расширенная постановка задачи

4. Реализация проектных решений

5. Заключение

6. Список литературы

7. Приложение (листинг программы)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель доцент / Святов К.В. /

должность подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ П.С. Кондратьев /

подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г

Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**ОТЗЫВ  
руководителя на курсовой проект (работу)**

Студента Кондратьева Павла Сергеевича

фамилия, имя и отчество

Факультет ФИСТ группа ИВТАСмд-11 курс 1

Дисциплина «Программная инженерия»

Тема проекта (работы) «Разработка программно доступных компонент для организации робототехнических соревнований в виртуальной среде»

Отмечаются следующие моменты: актуальность темы исследования; соответствие содержания и структуры курсовой работы ее теме; степень разработанности проблемы, наиболее интересно исследованные вопросы. Оценивается степень самостоятельности и инициативы студента; умение пользоваться различными источниками информации; уровень его теоретической подготовки; умение анализировать научные материалы, делать практические выводы; знание основных концепций, научной и специальной литературы по избранной теме. Содержится оценка проекта (работы) руководителем.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель доцент \_\_\_\_\_\_\_\_\_\_\_\_\_ / Святов К.В. /

должность, учёная степень, ученое звание подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

**Оглавление**

[Введение 5](#_Toc62499020)

[Расширенная постановка задачи 7](#_Toc62499021)

[Анализ основных видов проектных решений и инструментальных средств Webots API 8](#_Toc62499022)

[Реализация проектных решений 11](#_Toc62499023)

[Encoder 12](#_Toc62499024)

[Supervisor 16](#_Toc62499025)

[GPS 18](#_Toc62499026)

[Описание исходных кодов тестового задания и процесса их отладки 20](#_Toc62499027)

[Заключение 28](#_Toc62499028)

[Список используемых источников 29](#_Toc62499029)

[Приложение 30](#_Toc62499030)

[Листинг программы 30](#_Toc62499031)

# Введение

В современном мире область применения робототехники в различных сферах деятельности человека очень широкая и не перестает расти. Применение роботов позволяет значительно снизить участие человека в тяжелой и опасной работе. Например, работа в оборонных, химических, атомных сферах, тушение пожаров без помощи оператора, выполнение спасательных операций или передвижение по заранее неизвестной местности. Постепенно роботы входят и в обычную жизнь человека. Использование мобильных роботов позволяем удовлетворять ежедневные потребности: роботы-сиделки, роботы-нянечки, роботы-домработницы и т. д.. Как следствие современное общество очень нуждается в грамотных специалистах в этой области. В связи с этим обучение робототехнике детей становится все больше актуальной и значимой задачей.

Также робототехника может выступать не только как самостоятельный предмет, но и внедрятся в остальные школьные дисциплины. Робототехнические конструкторы можно использовать при демонстрации учебных экспериментов по физике, математике, физике и биологии, что позволяет увидеть картину реального мира.

При этом самой очевидной проблемой в проведении занятий по робототехнике является зависимость от реальных устройств. Количество доступных роботов как правило сильно ограниченно, а их цена может быть достаточно высока, что мешает широкому распространению робототехнических кружков.

В этой ситуации на помощь могут прийти робототехнические симуляторы. Они как правило представляют из себя ПО, в котором можно сконструировать собственного робота или выбрать робота из уже существующих вариантов. А также запрограммировать робота на языке программирования, доступным в платформе.

Но с робототехническими симуляторами также есть ряд проблем. Первая и основная проблема — это необходимость в установке ПО на личный компьютер, установка различных зависимостей и библиотек.

Целью данной курсовой работы является разработка программно доступных компонент для организации робототехнических соревнований в виртуальной среде. Это позволит обучающимся заниматься непосредственно программированием роботов на тестовых упражнениях в облачной среде.

Задачи курсовой работы:

1. Разработка сцены (мира робота)
2. Добавление различных моделей мобильного робота
3. Изучение возможностей supervisor API и GPS
4. Одометрия робота (определение текущей координаты)
5. С использованием стандартных средств Webots необходимо разработать контроллер для решения транспортной задачи

# Расширенная постановка задачи

Для дальнейшей разработки необходимо выбрать один из уже существующих робототехнических симуляторов.

Одним из наиболее известных симуляторов является Webots. Webots – это бесплатный симулятор трехмерного робота с открытым исходным кодом, используемый в промышленности, образовании и исследованиях.

Проект Webots стартовал в 1996 году и первоначально был разработан доктором Оливье Мишелем из Швейцарского федерального технологического института (EPFL) в Лозанне, Швейцария, а затем с 1998 года Cyberbotics Ltd. в качестве проприетарного лицензионного программного обеспечения. С декабря 2018 года он выпускается под бесплатной лицензией Apache 2 с открытым исходным кодом.

Webots – это симулятор роботов с открытым исходным кодом, выпущенный на условиях лицензии Apache 2.0. Он предоставляет полную среду разработки для моделирования, программирования и моделирования роботов, транспортных средств и биомеханических систем.

Webots включает в себя большую коллекцию свободно изменяемых моделей роботов, датчиков, исполнительных механизмов и объектов. Кроме того, также можно создавать новые модели с нуля или импортировать их. При разработке модели робота пользователь указывает как графические, так и физические свойства объектов. Графические свойства включают форму, размеры, положение и ориентацию, цвета и текстуру объекта. Физические свойства включают массу, коэффициент трения, а также постоянные пружины и демпфирования. В программе присутствует простая гидродинамика.

Webots включает в себя набор датчиков и исполнительных механизмов, часто используемых в роботизированных экспериментах, например лидары, радары, датчики приближения, датчики света, сенсорные датчики, GPS, акселерометры, камеры, излучатели и приемники, серводвигатели (вращательные и линейные), датчик положения и силы, светодиоды, захваты, гироскопы, компас, IMU и т. д.

Программы контроллера робота могут быть написаны вне Webots на языках C, C ++, Python, ROS, Java и MATLAB с использованием простого API.

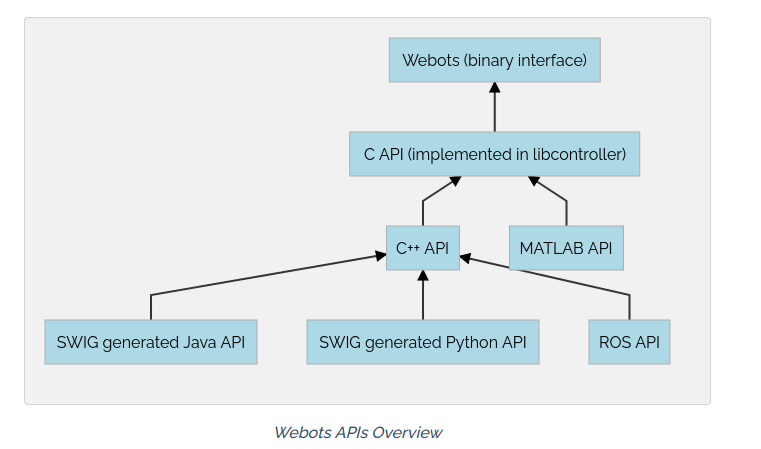
Webots предлагает возможность делать снимки экрана и записывать имитационные фильмы. Миры Webots хранятся в кроссплатформенных файлах. wbt, формат которых основан на языке VRML. Также можно импортировать и экспортировать миры или объекты Webots в формате VRML. Пользователи могут взаимодействовать с запущенной симуляцией в любое время, то есть можно перемещать роботов и другой объект с помощью мыши во время симуляции.

Большим преимуществом Webots по сравнению с другими симуляторами является наличие Web клиента.

## Анализ основных видов проектных решений и инструментальных средств Webots API

Классы и методы контроллеров C++, Java и Python являются объектно-ориентированными языками программирования, и поэтому соответствующие API Webots организованы в классы. Иерархия классов построена поверх API C и в настоящее время содержит около 25 классов и 200 методов (функций).

API Java и Python автоматически генерируются из API C++ с помощью SWIG. Поэтому имена классов и методов, а также количество параметров и их типы очень похожи в этих трех языках.



Соглашение об именовании классов и методов C++/Java/Python напрямую соответствует именам функций C API.

Реализация контроллера C++/Java/Python должна быть помещена в определяемый пользователем класс, производный от одного из классов Webots: Robotor или Supervisor. Класс контроллера должен быть производным от Robot класса, если supervisor поле узла робота есть FALSE, и от Supervisor класса, если оно есть TRUE.

Supervisor это подкласс Robotкласса. Следовательно, можно вызывать методы Robot's, такие как, например, функции steporget LED, из Supervisor контроллеров. Но вызвать Superviso rметоды из контроллера невозможно Robot. Например, невозможно будет вызвать worldReload функцию из Robot контроллера.

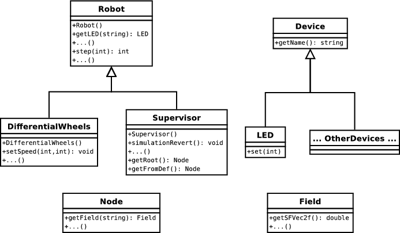


Рисунок 1. Небольшое подмножество Webots oriented-object API

Как правило, определяемый пользователем класс контроллера должен иметь run функцию, реализующую основной цикл контроллера. Этот цикл должен содержать вызов метода Robot's step. Затем единственная обязанность функции контроллера main состоит в том, чтобы создать экземпляр определяемого пользователем класса контроллера, вызвать его run метод и, наконец, удалить (только на языке C++) экземпляр: см. примеры ниже. Обратите внимание, что контроллер никогда не должен создавать более одного экземпляра производного класса, иначе результаты будут неопределенными.

Обратите внимание, что в отличие от API C, API C++/Java/Python не имеют (и не нуждаются) таких функций, как функции wb\_robot\_initandwb\_robot\_cleanup. Необходимые процедуры инициализации и очистки автоматически вызываются из конструктора и деструктора базового класса.

В C++ / Java/Python каждое устройство Webots реализовано как отдельный класс, есть DistanceSensor класс, TouchSensor класс, RotationalMotor класс и т. д. Различные экземпляры устройств могут быть получены с помощью специальных методов Robot класса, таких как функции getDistanceSensororgetTouchSensor. WbDeviceTagВ C++/Java/Python его нет.

# Реализация проектных решений

Одометрия (Odometry – от греческих слов hodos («перемещение», «путешествие») и metron («мера», «измерять»)) — использование данных о движении приводов, для оценки перемещения.

Одометрия применяется:

* в роботах (для подсчёта пройденного пути)
* станках с ЧПУ (для отслеживания положения инструмента),
* принтерах (для отслеживания положения печатающей головки),
* устройствах управления, например – компьютерных мышках (для отслеживания перемещения)

Одометрия не является методом определения положения, а лишь средством его оценки.

Стандартной схемой одометрии робота, является использование энкодеров, считывающих угол поворота колёс.

Энкодер – это устройство преобразующее линейное или угловое перемещение в последовательность сигналов, позволяющих определить величину перемещения.

Так же информацию о своем абсолютном положении из программы можно получить с помощью контроллера Supervisor и GPS. Для тестирования был выбран iRobot's Create – это настраиваемая рама, основанная на знаменитой платформе вакуумной очистки Roomba и производимая компанией iRobot.

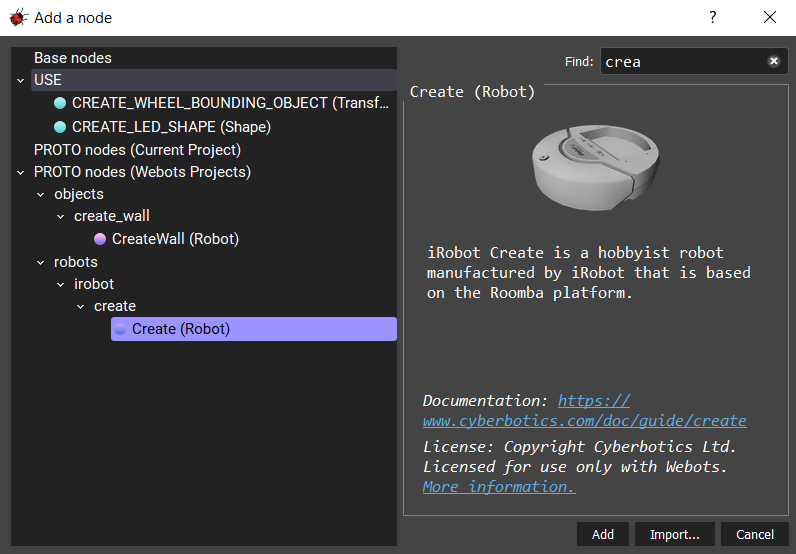


Рисунок 2. iRobot's Create робот

## Encoder

Position Sensor – это датчик, который облегчает измерение механического положения. Датчик положения может указывать абсолютное положение (положение) или относительное положение (перемещение) с точки зрения линейного перемещения, угла поворота или трехмерного пространства.

В контроллере робота создадим экземпляры для наших энкодеров которые в Webots являются датчиками положения, в отличие от двигателей, где мы устанавливаем положение и скорость для датчиков положения, мы включаем их с помощью отметка времени.

Датчик подключен к вращательному двигателю показания, которые мы получаем находятся в радианах обычно с помощью энкодера вы получаете тики энкодера, а затем добавляете эти тики энкодера, чтобы найти выясните, сколько вращалось ваше колесо, а затем на основе этого вы вычисляете, сколько ваш робот переместился в Webots с датчиком положения вычисление кодера уже было сделано для нас значение, которое мы получаем, выражено в радианах, оно означает величину вращения нашего колеса. итак, наш следующий шаг-вычислить, сколько вращения в радианах эквивалентно пройденному расстоянию.

Мы знаем, что один оборот колеса равен равный 6.28 радианам мы также знаем, что один оборот колеса означает всю окружность колесо покрыло плоскость, другими словами, 6,28 радиана эквивалентно окружности колеса в терминах пройденного расстояния, так что с помощью этого мы можем вычислить, что такое один радиан вращения эквивалентно и мы будем называть это блоком кодера, мы знаем радиус колеса, потому что данные были взяты с модель робота.

Инкрементные энкодеры, расположенные на колёсах робота, фиксируют «пройденное расстояние».

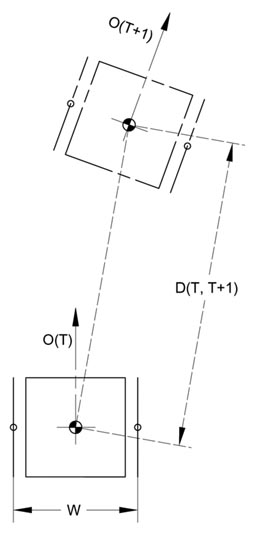


Рисунок 3. Новое положение центра O робота в момент времени t+1

, где

* O(t) – положение робота в момент времени t
* Dr – расстояние, пройденное правым колесом робота
* Dl – расстояние, пройденное левым колесом робота
* W – ширина робота

Расстояние, пройденное за этот промежуток времени можно оценить так:

Линейная аппроксимация криволинейной траектории (т.о., чем больше интервал времени между отсчётами – тем хуже приближение).

Для отображения координат робота на карте можно рассчитать его Декартовы координаты:

Т.о., одометрия – недорогое средство, которое даёт хорошую кратковременную точность и позволяет реализовать большую частоту дискретизации.

Однако, необходимо учитывать, что при рассмотрении более длительных промежутков времени, накопление погрешностей приводит к увеличению ошибки ориентации, которая растёт пропорционально пути, пройденному роботом.

**Источники погрешности:**

* погрешность измерения радиуса колёс
* различные размеры колёс (для роботов у которых больше одного колеса)
* ошибки подсчёта импульсов от энкодеров
* низкая частота обработки одометрии

while self.step(self.timeStep) != 1:

self.ps\_values[0] = self.left\_ps.getValue()

self.ps\_values[1] = self.right\_ps.getValue()

print(---------------------------------\‘n'position sensor values: {} {}'.format(self.ps\_values[0], self.ps\_values[1]))

for ind in range(2):

self.diff = self.ps\_values[ind] - self.last\_ps\_values[ind]

if self.diff < 0.001:

self.diff = 0

self.ps\_values[ind] = self.last\_ps\_values[ind]

self.dist\_values[ind] = self.ps\_values[ind] \* self.encoder\_unit

print('distance values: {} {}'.format(self.dist\_values[0], self.dist\_values[1]))

# compute linear and angular velocity for robot

self.v = (self.dist\_values[0] + self.dist\_values[1]) / 2.0

self.w = (self.dist\_values[0] - self.dist\_values[1]) / self.distance\_between\_weels

self.robot\_pos[2] += (self.w \* self.dt)

self.vx = self.v \* math.cos(self.robot\_pos[2])

self.vy = self.v \* math.sin(self.robot\_pos[2])

self.robot\_pos[0] += self.vx

self.robot\_pos[1] += self.vy

print('robot\_pose: {}'.format(self.robot\_pos))

for ind in range(2):

self.last\_ps\_values[ind] = self.ps\_values[ind]

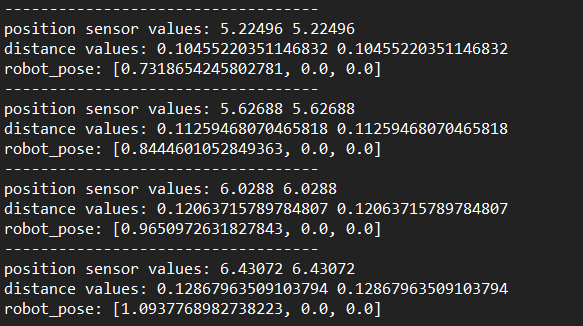


Рисунок 4. Данные с PositionSersor при линейном движение

## Supervisor

Это не узел, а набор функций, доступных для каждого из них Узел робота, чей supervisor поле имеет значение TRUE. Супервизор API может использоваться для доступа к дополнительным функциям, недоступным обычному пользователю. Робот.

Одна важная вещь, которую следует иметь в виду, заключается в том, что API супервизора соответствует функциям, которые обычно недоступны на реальных роботах; он скорее соответствует человеческому вмешательству в экспериментальную установку. Следовательно, wb\_robot\_\*wb\_supervisor\_\* различие vs намеренно и направлено на то, чтобы напомнить пользователю, что функции супервизора API не могут быть легко перенесены на реальных роботов.

Нам нужно отслеживать позицию робота. Это можно сделать с помощью:

* функции getSFVec3f(), которая возвращает список из 3 координат.
* функции getPosition(), возвращает положение узла, выраженное в глобальной (мировой) системе координат. Эта функция возвращает вектор, содержащий ровно 3 значения.

Супервизор API часто используется для записи траекторий роботов. Конечно, робот может найти свое местоположение с помощью GPS, но, когда необходимо отслеживать нескольких роботов одновременно и централизованно, гораздо проще использовать Супервизор API.

TIME\_STEP = 64

robot\_coords = [[0, 0, 0], [0, 0, 0, 0]]

supervisor = Supervisor()

keyboard = Keyboard()

keyboard.enable(TIME\_STEP)

emitter = supervisor.getDevice("emitter")

# do this once only

robot\_node = supervisor.getFromDef("E-puck")

if robot\_node is None:

sys.stderr.write("No DEF E-puck node found in the current world file\n")

sys.exit(1)

trans\_field\_translation = robot\_node.getField('translation')

trans\_field\_rotation = robot\_node.getField('rotation')

print('Press S to read the Supervisors position and rotation\n')

while supervisor.step(TIME\_STEP) != -1:

# this is done repeatedly

# get handle to robot's translation and rotation field

translation = trans\_field\_translation.getSFVec3f()

rotation = trans\_field\_rotation.getSFRotation()

robot\_coords[0] = translation

robot\_coords[1] = rotation

if(keyboard.getKey() == ord('S')):

print('translation:', translation)

print('rotation:', rotation)

message = struct.pack('>7f', robot\_coords[0][0], robot\_coords[0][1], robot\_coords[0][2], robot\_coords[1][0], robot\_coords[1][1], robot\_coords[1][2], robot\_coords[1][3])

emitter.send(message)

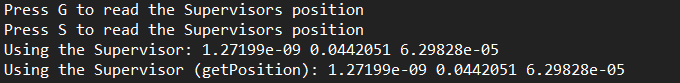


Рисунок 5. Данные с Supervisor

Данный код реализован в отдельной Base Node (Узел робота) – это универсальный тип робота. В нем можно реализовать Emitter который может отправлять данные, но не может принимать их. Чтобы имитировать однонаправленную связь между двумя роботами, один робот должен иметь излучатель, а другой - приемник. Чтобы имитировать двунаправленную связь между двумя роботами, каждый робот должен иметь как излучатель, так и приемник. Обратите внимание, что сообщения никогда не передаются от одного робота к другому.

## GPS

Узел GPS используется для моделирования датчика глобального позиционирования (GPS), который может получать информацию о своем абсолютном положении из программы контроллера.

**Включение, отключение и считывание GPS-измерений**

* wb\_gps\_enable функция позволяет пользователю включить GPS-измерения. sampling\_period Аргумент задает период дискретизации датчика и выражается в миллисекундах. Обратите внимание, что первое измерение будет доступно только по истечении первого периода отбора проб.
* wb\_gps\_get\_values функция возвращает текущее измерение GPS. Значения возвращаются в виде 3D-вектора, поэтому для доступа к вектору допустимы только индексы 0, 1 и 2. Возвращенный вектор показывает абсолютное положение устройства GPS. Эта позиция может быть выражена либо в декартовой системе координат Webots, либо с использованием широты-долготы-высоты, в зависимости от значения gpsCoordinateSystemполя узла WorldInfo.
* Примечание [C, C++]: возвращаемый вектор является указателем на внутренние значения, управляемые узлом GPS, поэтому освобождение этого указателя незаконно. Кроме того, обратите внимание, что указанные значения действительны только до следующего вызова функций wb\_robot\_steporRobot::step. Если эти значения необходимы для более длительного периода, они должны быть скопированы.

def run(self):

print('Press G to read the Supervisors position\n')

print('Press S to read the Supervisors position\n')

while self.step(self.timeStep) != -1:

key\_down = self.keyboard.getKey()

if(key\_down == ord('G')):

gps\_values = self.gps.getValues()

print('Irobot using the GPS: %g %g %g' % (gps\_values[0], gps\_values[1], gps\_values[2]))

# main Python program

controller = MyController()

controller.run()

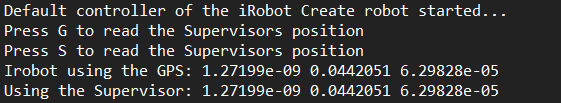


Рисунок 6. Данные с GPS

Данный код реализован в самом iRobot's Create, с добавлением Node GPS. На Python getValues функция возвращает 3D-вектор в виде списка, содержащего три поплавка.

## Описание исходных кодов тестового задания и процесса их отладки

Создадим новое тестовое задание по образцу тех упражнений, которые уже существуют в системе.

Создадим сцену для упражнения. С помощью project wizard new project directory создадим новый проект и при создании добавим инициализацию пустой сцены. При этом создастся следующая сцена:

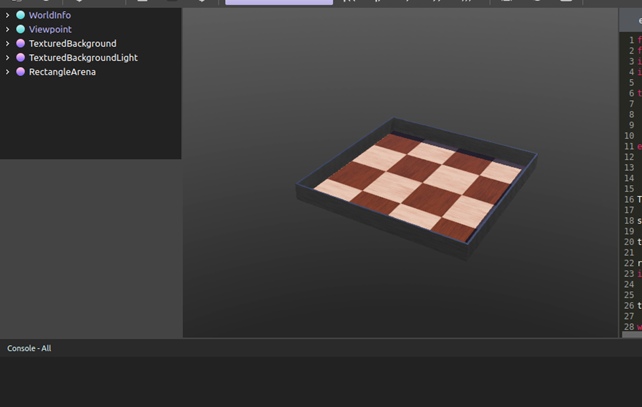


Рисунок 7. Пустая сцена

Теперь добавим на сцену робота. Для нашего упражнения будет достаточно учебного робота E-puck.

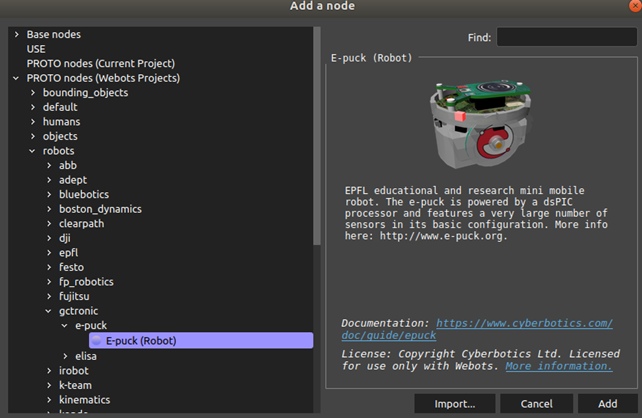


Рисунок 8. E-puck робот

Добавим этого робота на сцену. С помощью project wizard new robot controller создадим новый контроллер и привяжем его к роботу, выбрав нужный язык программирования. После чего привязать созданный контроллер к текущему роботу в дереве решений созданного проекта (робот – E-puck).

Для создания <Руководителя> в проекте нужно добавить новый узел робота, к которому подключим Супервизор. Это по большей части не узел, а набор функций, доступных для каждого узла Робота, поле супервизора которого имеет значение TRUE. API Супервизора можно использовать для доступа к дополнительным функциям, недоступным обычному роботу.

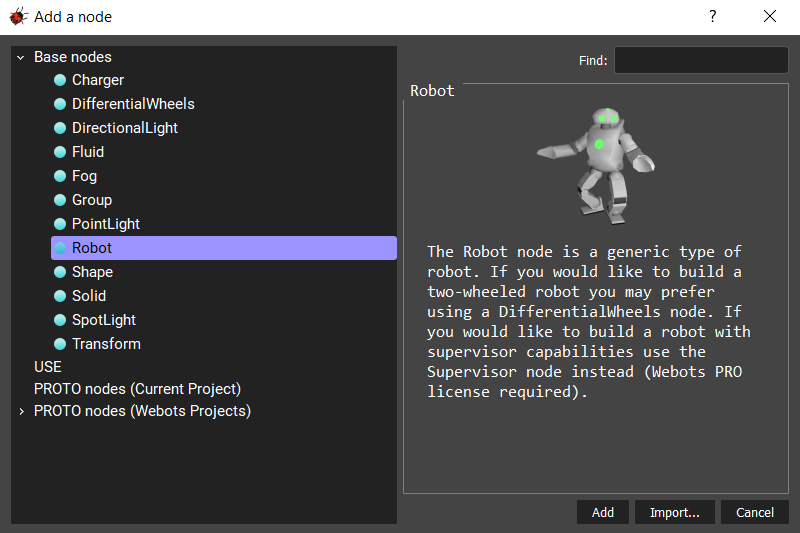


Рисунок 9. Создания Узла робота

После создания узла, нужно сделать пару приготовлений перед его работой:

1. Дать название в поле DEF
2. Включить возможность использования Супервизора
3. Добавить Узел эмиттера, который должен быть добавлен к дочерним элементам робота или супервизора
4. Добавить контроллер

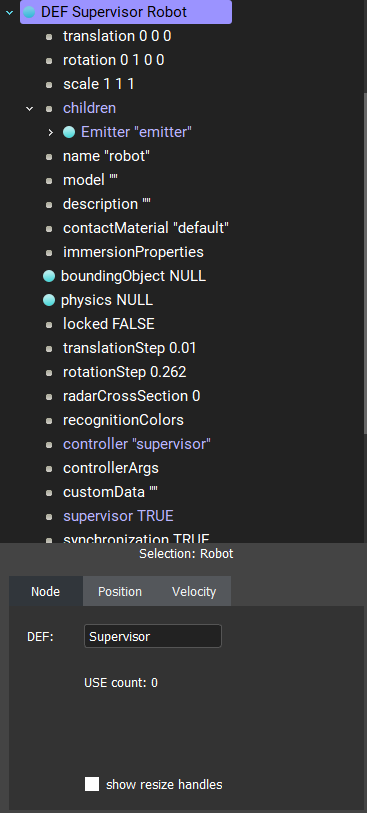


Рисунок 10. Дерево решений

**Примечание:** На узел, который назван с помощью ключевого слова DEF, можно ссылаться позже по его имени в том же файле с инструкциями USE. Ключевые слова DEF и USE можно использовать для уменьшения избыточности в файлах ".wbt" и ".proto". Область действия имени DEF ограничена одним файлом ".wbt" или ".proto". Если нескольким узлам дано одно и то же имя DEF, то каждый оператор USE ссылается на ближайший узел с заданным именем DEF, предшествующим ему в файле ".wbt" или ".proto".

Это позволить нам запросто получать любые параметры из узла WorldInfo, который предоставляет общую информацию о моделируемом мире. Поэтому с помощью Супервизора можно получить доступа к окну дерева сцен.

Ниже на рисунке 11 можно увидеть в коде контроллера, на строке 16, как с помощью supervisor.getFromDef("E-puck"), получаем сам узел робота, со всеми его параметрами, для дальнейшего использования.

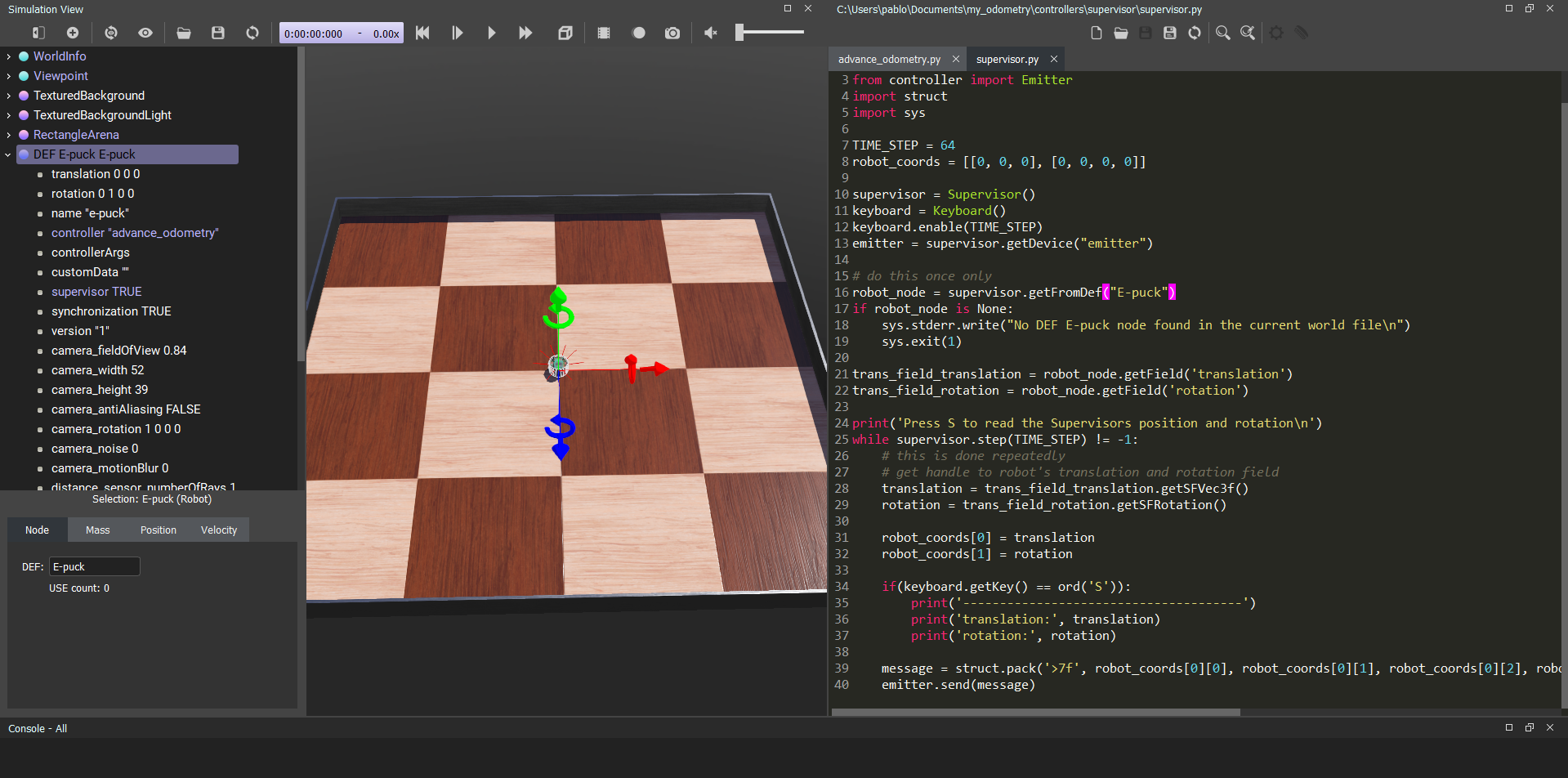


Рисунок 11. Узел Супервизора

Для предоставления данных из Супервизора другим робота используется узел излучателя (Emitter), который отправляет сигнал, который в свою очередь может принять приёмник (Receiver).

Сам Emitter имеет поля:

* Type – тип сигналов: "радио", "последовательный " или"инфракрасный". Сигналы типа "радио" (по умолчанию) и "последовательный" передаются без учета препятствий. Однако сигналы типа "инфракрасный" действительно учитывают потенциальные препятствия между излучателем и приемником.
* Range – радиус сферы излучения (в метрах). Приемник может принимать сообщение только в том случае, если он находится внутри сферы излучения. Значение -1 (по умолчанию) для rangeсчитается бесконечным диапазоном.
* Aperture – угол раскрытия конуса излучения (в радианах); только для" инфракрасного " излучения. Вершина конуса расположена в начале координат ([0 0 0]) системы координат эмиттера, а ось конуса совпадает с осью z системы координат эмиттера.
* Channel – канал передачи. Это идентификационный номер для "инфракрасного" излучателя или частота для "радио" излучателя. Обычно приемник должен использовать тот же канал, что и излучатель для приема излучаемых данных. Однако специальный канал -1 позволяет транслировать сообщения по всем каналам. Канал 0 (по умолчанию) зарезервирован для связи с физическим плагином.

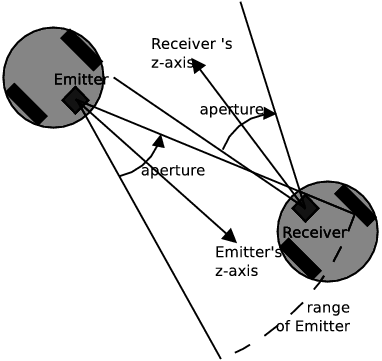


Рисунок 12. Иллюстрация апертуры и дальности действия инфракрасного излучателя / приемника

Благодаря полю • Channel мы можем подписать на данный канал и получать данные для различных устройств или же ограничивая доступ входящих пакетов данных.

В свою очередь Receiver запускает прослушивание приемником входящих пакетов данных. Прием данных активируется в фоновом режиме контура контроллера со скоростью один раз sampling\_periodв секунду (выражается в миллисекундах). Входящие пакеты данных добавляются в хвост очереди приема (см. Этот рисунок). Входящие пакеты данных будут отброшены, если размер буфера приемника (указанный в узле приемника) будет превышен. Чтобы избежать переполнения буфера, программа контроллера должна считывать пакеты данных с достаточно высокой скоростью. sampling\_periodАргумент указывает период дискретизации приемника и выражается в миллисекундах. Узел-приемник принимает и ставит в очередь входящие пакеты, так как он включен, но первые пакеты данных могут быть извлечены только после истечения первого периода выборки.

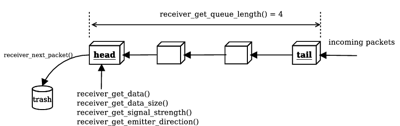


Рисунок 13. Очередь пакетов получателя

По полученным данным из Супервизора можно знать текущие координаты и угол поворота робота на нашей сцене. Это позволит написать простой код для перемещения робота из текущей координаты в заданную координату мира симуляции.

Основной логикой движения робота является: Сравнение текучей ориентации с фактической ориентации робота. После чего робот поворачивается на нужный угол до цели и начинает движение. По достижению цели робот должен остановиться и дать знать то что он добрался до заданной цели (в качестве примера робот должен добраться до точки 0.5,0.5).

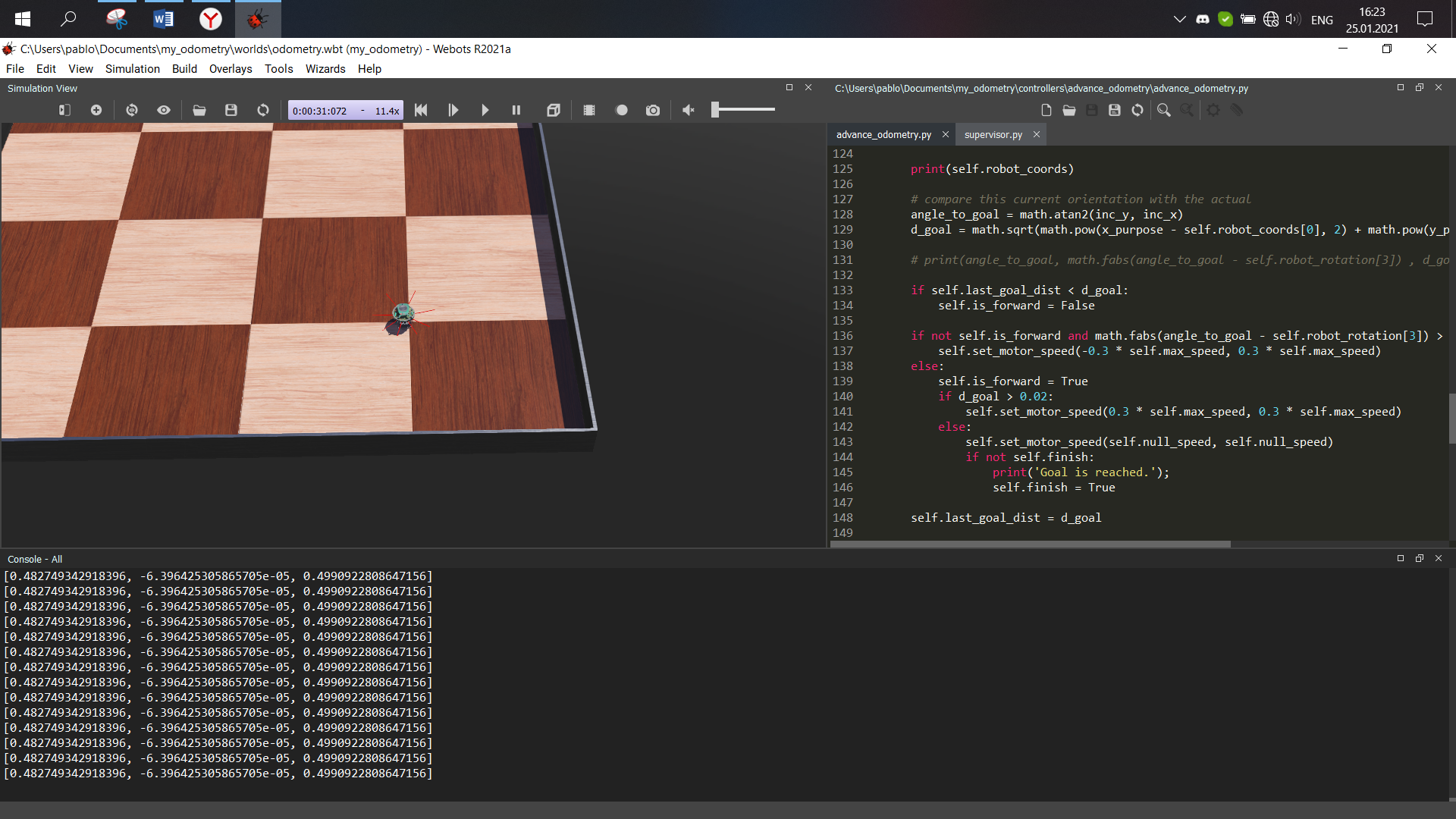


Рисунок 14. Пример движения робота до цели

# Заключение

В данной курсовой работе был рассмотрен робототехнический симулятор Webots, а также механизмы работы его узлов и функций работы API.

Был рассмотрен запуск тренировочного задания, взятого из репозитория проекта Webots. Так же было разработано собственное тестовое упражнение, на примере которого были рассмотрены различные варианты одометрии робота в виртуальной среде.

Также был рассмотрен узел робота, supervisor, поле которого задано TRUE, может использоваться как любой обычный узел робота, но, кроме того, supervisor функции также могут использоваться для управления процессом моделирования и изменения дерева сцен. Например, Супервизор AP может заменить человеческие действия, такие как измерение расстояния, пройденного роботом, или перемещение его обратно в исходное положение и т. д.

# Список используемых источников

1. Ивкина К. И. Актуальность обучения робототехнике / К. И. Ивкина, Л. М. Ивкина// Актуальные проблемы авиации и космонавтики. – 2015. – С. 942 – 943.
2. Документация симулятора Webots [Электронный ресурс]. – Режим доступа: <https://www.cyberbotics.com/doc>
3. Движение робота к точке с заданными координатами [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/277829/
4. Одометрия [Электронный ресурс]. – Режим доступа: <http://robocraft.ru/blog/technology/736.html>
5. Feedback & Odometry: Coursera’s Control of Mobile Robots with ROS and ROSbots — Part 3 [Электронный ресурс]. – Режим доступа: <https://hackernoon.com/feedback-odometry-courseras-control-of-mobile-robots-with-ros-and-rosbots-part-3-e9d8e4df6df1>
6. Управление точностью движений робота. Движение на заданное расстояние. [Электронный ресурс]. – <https://www.sites.google.com/site/pervyesagivrobototehniku/home/zanatie13-14>
7. A Clever Trigonometry Based Controller [Электронный ресурс]. – Режим доступа: http://faculty.salina.k-state.edu /tim/robotics\_sg/Control/ controllers/trig\_trick.html
8. Навигация. Первые шаги. Курс, пеленг. [Электронный ресурс]. – Режим доступа: https://navlib.net/navlib-navigation-hdg-brg/

# Приложение

## Листинг программы

**Supervisor Controller**

from controller import Supervisor

from controller import Keyboard

from controller import Emitter

import struct

import sys

TIME\_STEP = 64

robot\_coords = [[0, 0, 0], [0, 0, 0, 0]]

supervisor = Supervisor()

keyboard = Keyboard()

keyboard.enable(TIME\_STEP)

robot\_node = supervisor.getFromDef("E-puck")

if robot\_node is None:

sys.stderr.write("No DEF E-puck node found in the current world file\n")

sys.exit(1)

emitter = supervisor.getDevice("emitter")

trans\_field\_translation = robot\_node.getField('translation')

trans\_field\_rotation = robot\_node.getField('rotation')

print('Press S to read the Supervisors position and rotation\n')

while supervisor.step(TIME\_STEP) != -1:

translation = trans\_field\_translation.getSFVec3f()

rotation = trans\_field\_rotation.getSFRotation()

robot\_coords[0] = translation

robot\_coords[1] = rotation

if(keyboard.getKey() == ord('S')):

print('translation: %g %g %g' % (translation[0], translation[1], translation[2]))

print('rotation: %g %g %g' % (rotation[0], rotation[1], rotation[2]))

message = struct.pack('>7f', robot\_coords[0][0], robot\_coords[0][1], robot\_coords[0][2], robot\_coords[1][0], robot\_coords[1][1], robot\_coords[1][2], robot\_coords[1][3])

emitter.send(message)

**Gps и Receiver для Supervisor**

from controller import Robot

from controller import GPS

from controller import Keyboard

import struct

cliff\_sensors\_name = ['cliff\_left', 'cliff\_front\_left', 'cliff\_front\_right', 'cliff\_right']

class MyController(Robot):

def \_\_init\_\_(self):

super(MyController, self).\_\_init\_\_()

self.timeStep = 64 # set the control time step

self.null\_speed = 0

self.max\_speed = 6.28

self.cliff\_sensors = []

self.robot\_coords = []

self.robot\_rotation = []

self.gps = GPS('gps')

self.keyboard = Keyboard()

self.receiver = self.getDevice('receiver')

# get device tags

self.keyboard.enable(self.timeStep)

self.gps.enable(self.timeStep)

self.receiver.enable(self.timeStep)

for i in range(0, len(cliff\_sensors\_name)):

self.cliff\_sensors.append(self.getDevice(cliff\_sensors\_name[i]))

self.cliff\_sensors[i].enable(self.timeStep)

self.leftMotor = self.getDevice('left wheel motor')

self.rightMotor = self.getDevice('right wheel motor')

self.leftMotor.setPosition(float('inf'))

self.rightMotor.setPosition(float('inf'))

self.leftMotor.setVelocity(self.null\_speed)

self.rightMotor.setVelocity(self.null\_speed)

self.left\_position\_sensor = self.getDevice('left wheel sensor')

self.right\_position\_sensor = self.getDevice('right wheel sensor')

self.left\_position\_sensor.enable(self.timeStep)

self.right\_position\_sensor.enable(self.timeStep)

print('Default controller of the iRobot Create robot started...\n');

def go\_forward(self, left\_percent, right\_percent):

self.leftMotor.setVelocity(left\_percent \* self.max\_speed)

self.rightMotor.setVelocity(right\_percent \* self.max\_speed)

def stop(self):

self.leftMotor.setVelocity(self.null\_speed)

self.rightMotor.setVelocity(self.null\_speed)

def get\_supervisor\_data(self):

if(self.receiver.getQueueLength() == 0):

return

while(self.receiver.getQueueLength() > 1):

self.receiver.nextPacket()

buffer = self.receiver.getData()

buffer\_data = list(struct.unpack('>7f', buffer))

self.robot\_coords = buffer\_data

self.robot\_rotation = buffer\_data[3:]

def run(self):

print('Press G to read the Supervisors position\n')

print('Press S to read the Supervisors position\n')

print('Press D to read the Supervisors position and rotation\n')

while self.step(self.timeStep) != -1:

key\_down = self.keyboard.getKey()

if(key\_down == ord('G')):

gps\_values = self.gps.getValues()

print('Irobot using the GPS: %g %g %g' % (gps\_values[0], gps\_values[1], gps\_values[2]))

if(key\_down == ord('D')):

self.get\_supervisor\_data()

print('coords', self.robot\_coords)

self.leftMotor.setVelocity(-self.max\_speed)

self.rightMotor.setVelocity(self.max\_speed)

# main Python program

controller = MyController()

controller.run()

**Odometry Controller**

from controller import Robot

import math

import struct

class MyController(Robot):

def \_\_init\_\_(self):

super(MyController, self).\_\_init\_\_()

self.timeStep = 64 # set the control time step

self.null\_speed = 0

self.max\_speed = 6.28

self.speed = [0, 0]

self.wheel\_radius = 0.02

self.distance\_between\_weels = 0.05685

self.diff = 0

self.v = 0

self.w = 0

self.vx = 0

self.vy = 0

self.distance\_sensor = []

self.dist\_sensor\_values = [0, 0, 0, 0, 0, 0, 0, 0]

self.robot\_pos = [0, 0, 0]

self.last\_ps\_values = [0, 0]

self.ps\_values = [0, 0]

self.dist\_values = [0, 0]

self.last\_goal\_dist = math.inf

self.robot\_coords = [0, 0, 0]

self.robot\_rotation = [0, 0, 0, 0]

self.left\_obstacle = False

self.right\_obstacle = False

self.is\_forward = False

self.finish = False

self.weel\_cirum = 2 \* math.pi \* self.wheel\_radius

self.encoder\_unit = self.weel\_cirum / 6.28

# get device tags

self.left\_motor = self.getDevice('left wheel motor')

self.right\_motor = self.getDevice('right wheel motor')

self.left\_motor.setPosition(float('inf'))

self.left\_motor.setVelocity(self.null\_speed)

self.right\_motor.setPosition(float('inf'))

self.right\_motor.setVelocity(self.null\_speed)

self.left\_ps = self.getDevice('left wheel sensor')

self.left\_ps.enable(self.timeStep)

self.right\_ps = self.getDevice('right wheel sensor')

self.right\_ps.enable(self.timeStep)

for ind in range(8):

self.distance\_sensor.append(self.getDevice('ps' + str(ind)))

self.distance\_sensor[ind].enable(self.timeStep)

self.gyro = self.getDevice('gyro')

self.gyro.enable(self.timeStep)

self.receiver = self.getDevice('receiver')

self.receiver.enable(self.timeStep)

print('Default controller of the E-puck robot started...\n');

def set\_motor\_speed(self, left, right):

""" Set up the motor speeds

Parameters

----------

left : float

Current values of the speed, acceleration, and torque/force of the left motor

right : float

Current values of the speed, acceleration, and torque/force of the right motor

"""

self.left\_motor.setVelocity(left)

self.right\_motor.setVelocity(right)

def get\_supervisor\_data(self):

""" Get the current robot coordinates from the supervisor

"""

if(self.receiver.getQueueLength() == 0):

print('not data')

return

while(self.receiver.getQueueLength() > 1):

self.receiver.nextPacket()

buffer = self.receiver.getData()

buffer\_data = list(struct.unpack('>7f', buffer))

self.robot\_coords = buffer\_data[:3]

self.robot\_rotation = buffer\_data[3:]

# print('-----------------------------------')

# print('robot\_coords', self.robot\_coords)

# print('robot\_rotation', self.robot\_rotation)

def obstacle\_avoidance(self):

""" Сhecking the proximity sensor for a remote obstacle

"""

self.left\_obstacle = self.dist\_sensor\_values[5] > 80.0 or self.dist\_sensor\_values[6] > 80.0 or self.dist\_sensor\_values[7] > 80.0

self.right\_obstacle = self.dist\_sensor\_values[0] > 80.0 or self.dist\_sensor\_values[1] > 80.0 or self.dist\_sensor\_values[2] > 80.0

print('left', self.left\_obstacle, 'right', self.right\_obstacle)

def go\_to\_position(self, x\_purpose, y\_purpose):

""" Move a Robot to a Certain Point

Parameters

----------

x\_purpose : float

target value of the x coordinate to the target

y\_purpose : float

target value of the y coordinate to the target

"""

inc\_x = self.robot\_coords[0] - x\_purpose

inc\_y = self.robot\_coords[2] - y\_purpose

# compare this current orientation with the actual

angle\_to\_goal = math.atan2(inc\_y, inc\_x)

d\_goal = math.sqrt(math.pow(x\_purpose - self.robot\_coords[0], 2) + math.pow(y\_purpose - self.robot\_coords[2], 2))

print(angle\_to\_goal, math.fabs(angle\_to\_goal - self.robot\_rotation[3]) , d\_goal, self.last\_goal\_dist)

if self.last\_goal\_dist < d\_goal:

self.is\_forward = False

if not self.is\_forward and math.fabs(angle\_to\_goal - self.robot\_rotation[3]) > 0.1:

self.set\_motor\_speed(-0.3 \* self.max\_speed, 0.3 \* self.max\_speed)

else:

self.is\_forward = True

if d\_goal > 0.02:

self.set\_motor\_speed(0.3 \* self.max\_speed, 0.3 \* self.max\_speed)

else:

self.set\_motor\_speed(self.null\_speed, self.null\_speed)

if not self.finish:

print('Goal is reached.');

self.finish = True

self.last\_goal\_dist = d\_goal

def run(self):

""" Running a simulator whose process runs in an infinite loop until it is terminated by webbots or by stopping the simulation

"""

while self.step(self.timeStep) != 1:

for ind in range(8):

self.dist\_sensor\_values[ind] = self.distance\_sensor[ind].getValue()

self.ps\_values[0] = self.left\_ps.getValue()

self.ps\_values[1] = self.right\_ps.getValue()

# print('-----------------------------------')

# print('position sensor values: {} {}'.format(self.ps\_values[0], self.ps\_values[1]))

# now with this we can compute the distance traveled

for ind in range(2):

self.diff = self.ps\_values[ind] - self.last\_ps\_values[ind]

if self.diff < 0.001:

self.diff = 0

self.ps\_values[ind] = self.last\_ps\_values[ind]

self.dist\_values[ind] = self.ps\_values[ind] \* self.encoder\_unit

# print('distance values: {} {}'.format(self.dist\_values[0], self.dist\_values[1]))

# compute linear and angular velocity for robot

self.v = (self.dist\_values[0] + self.dist\_values[1]) / 2.0

self.w = (self.dist\_values[0] - self.dist\_values[1]) / self.distance\_between\_weels

self.robot\_pos[2] += self.w

self.vx = self.v \* math.cos(self.robot\_pos[2])

self.vy = self.v \* math.sin(self.robot\_pos[2])

self.robot\_pos[0] += (self.vx \* self.dt)

self.robot\_pos[1] += (self.vy \* self.dt)

# print('robot\_pose: {}'.format(self.robot\_pos))

self.get\_supervisor\_data();

self.go\_to\_position(0.5, 0.5)

for ind in range(2):

self.last\_ps\_values[ind] = self.ps\_values[ind]

controller = MyController()

controller.run()