МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра «Вычислительные системы и технологии»

ОТЧЁТ

По лабораторной работе №2

по дисциплине «Аппаратное и программное обеспечение   
роботизированных систем»

«Знакомство с симулятором Webots»

ПРОВЕРИЛ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гай В.Е.

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Семенова Д.С.

**17-В-1**

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород

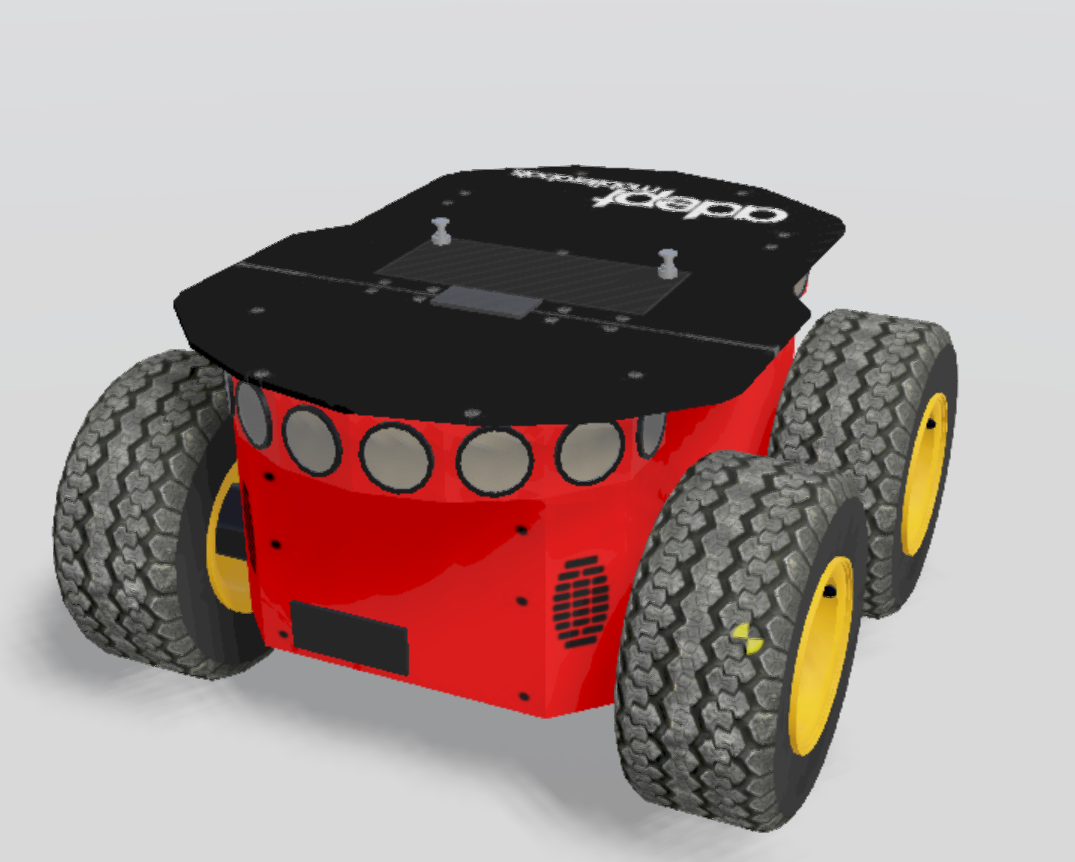
2020

**Цель:** познакомиться с симулятором Webots. Реализовать для конкретного робота (по вариантам) программу движения, а также создать новый мир и добавить в него объекты.

**Вариант:** pioneer3at.

**О роботе:**

Эта четырехколесная модель содержит 16 датчиков расстояний (8 на передней облицовке, 8 на задней облицовке) для измерений близости объектов. Робот Pioneer 3-AT представляет собой универсальную наружную базу, используемую для исследований и прототипирования приложений, включающих картографирование, навигацию, мониторинг, рекогносцировку и др.



**Мир.** Реализованный мир представлен ниже, а также к данному отчету приложен: видео – пример моделирования, а также сам созданный симулятор - мир в формате данного симулятора.



**Текст контроллера робота:**

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <webots/lidar.h>

#include <webots/motor.h>

#include <webots/robot.h>

#define TIME\_STEP 32

#define MAX\_SPEED 6.4

#define CRUISING\_SPEED 5.0

#define OBSTACLE\_THRESHOLD 0.1

#define DECREASE\_FACTOR 0.9

#define BACK\_SLOWDOWN 0.9

// gaussian function

double gaussian(double x, double mu, double sigma) {

return (1.0 / (sigma \* sqrt(2.0 \* M\_PI))) \* exp(-((x - mu) \* (x - mu)) / (2 \* sigma \* sigma));

}

int main(int argc, char \*\*argv) {

// init webots stuff

wb\_robot\_init();

// get devices

WbDeviceTag lms291 = wb\_robot\_get\_device("Sick LMS 291");

WbDeviceTag front\_left\_wheel = wb\_robot\_get\_device("front left wheel");

WbDeviceTag front\_right\_wheel = wb\_robot\_get\_device("front right wheel");

WbDeviceTag back\_left\_wheel = wb\_robot\_get\_device("back left wheel");

WbDeviceTag back\_right\_wheel = wb\_robot\_get\_device("back right wheel");

// init lms291

wb\_lidar\_enable(lms291, TIME\_STEP);

const int lms291\_width = wb\_lidar\_get\_horizontal\_resolution(lms291);

const int half\_width = lms291\_width / 2;

const int max\_range = wb\_lidar\_get\_max\_range(lms291);

const double range\_threshold = max\_range / 20.0;

const float \*lms291\_values = NULL;

// init braitenberg coefficient

double \*const braitenberg\_coefficients = (double \*)malloc(sizeof(double) \* lms291\_width);

int i, j;

for (i = 0; i < lms291\_width; ++i)

braitenberg\_coefficients[i] = gaussian(i, half\_width, lms291\_width / 5);

// init motors

wb\_motor\_set\_position(front\_left\_wheel, INFINITY);

wb\_motor\_set\_position(front\_right\_wheel, INFINITY);

wb\_motor\_set\_position(back\_left\_wheel, INFINITY);

wb\_motor\_set\_position(back\_right\_wheel, INFINITY);

// init speed for each wheel

double back\_left\_speed = 0.0, back\_right\_speed = 0.0;

double front\_left\_speed = 0.0, front\_right\_speed = 0.0;

wb\_motor\_set\_velocity(front\_left\_wheel, front\_left\_speed);

wb\_motor\_set\_velocity(front\_right\_wheel, front\_right\_speed);

wb\_motor\_set\_velocity(back\_left\_wheel, back\_left\_speed);

wb\_motor\_set\_velocity(back\_right\_wheel, back\_right\_speed);

// init dynamic variables

double left\_obstacle = 0.0, right\_obstacle = 0.0;

// control loop

while (wb\_robot\_step(TIME\_STEP) != -1) {

// get lidar values

lms291\_values = wb\_lidar\_get\_range\_image(lms291);

// apply the braitenberg coefficients on the resulted values of the lms291

// near obstacle sensed on the left side

for (i = 0; i < half\_width; ++i) {

if (lms291\_values[i] < range\_threshold) // far obstacles are ignored

left\_obstacle += braitenberg\_coefficients[i] \* (1.0 - lms291\_values[i] / max\_range);

// near obstacle sensed on the right side

j = lms291\_width - i - 1;

if (lms291\_values[j] < range\_threshold)

right\_obstacle += braitenberg\_coefficients[i] \* (1.0 - lms291\_values[j] / max\_range);

}

// overall front obstacle

const double obstacle = left\_obstacle + right\_obstacle;

// compute the speed according to the information on

// obstacles

if (obstacle > OBSTACLE\_THRESHOLD) {

const double speed\_factor = (1.0 - DECREASE\_FACTOR \* obstacle) \* MAX\_SPEED / obstacle;

front\_left\_speed = speed\_factor \* left\_obstacle;

front\_right\_speed = speed\_factor \* right\_obstacle;

back\_left\_speed = BACK\_SLOWDOWN \* front\_left\_speed;

back\_right\_speed = BACK\_SLOWDOWN \* front\_right\_speed;

} else {

back\_left\_speed = CRUISING\_SPEED;

back\_right\_speed = CRUISING\_SPEED;

front\_left\_speed = CRUISING\_SPEED;

front\_right\_speed = CRUISING\_SPEED;

}

// set actuators

wb\_motor\_set\_velocity(front\_left\_wheel, front\_left\_speed);

wb\_motor\_set\_velocity(front\_right\_wheel, front\_right\_speed);

wb\_motor\_set\_velocity(back\_left\_wheel, back\_left\_speed);

wb\_motor\_set\_velocity(back\_right\_wheel, back\_right\_speed);

// reset dynamic variables to zero

left\_obstacle = 0.0;

right\_obstacle = 0.0;

}

free(braitenberg\_coefficients);

wb\_robot\_cleanup();

return 0;

}

**Выводы:** в рамках выполнения данной лабораторной работы был опробован симулятор Webots, написан контроллер для робота, а также изучены некоторые главы документации и видео-уроки.