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:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ильин М.А.

**17-В-1**

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

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

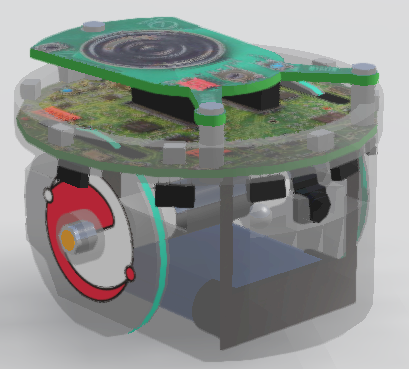
Нижний Новгород

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:** **робот** GCTronic' e-puck

**О роботе:**

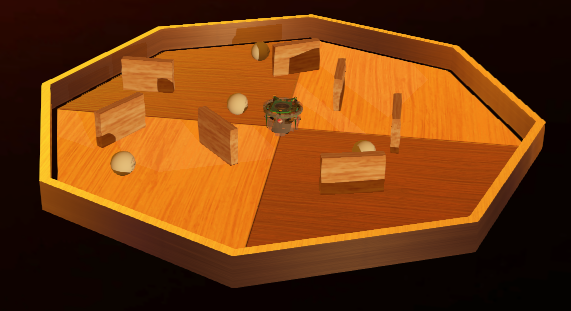
E-puck - это миниатюрный мобильный робот, первоначально разработанный в EPFL для учебных целей разработчиками успешного робота Khepera. Аппаратное и программное обеспечение e-puck полностью с открытым исходным кодом, обеспечивает низкий уровень доступа к каждому электронному устройству и предлагает неограниченные возможности расширения.



E-puck был разработан с учетом следующих требований:

* *Элегантный дизайн*
* *Гибкость*
* *Программное обеспечение для моделирования*
* *Удобство в использовании*
* *Надежность и техническое обслуживание*
* *Доступность*

**Мир.** Реализованный мир представлен ниже, а также к данному отчету приложен: видео – пример моделирования, а также файл виртуального пространства в формате данного симулятора.



**Текст контроллера робота:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <webots/device.h>

#include <webots/distance\_sensor.h>

#include <webots/led.h>

#include <webots/motor.h>

#include <webots/nodes.h>

#include <webots/robot.h>

/\* Device stuff \*/

#define DISTANCE\_SENSORS\_NUMBER 8

static WbDeviceTag distance\_sensors[DISTANCE\_SENSORS\_NUMBER];

static double distance\_sensors\_values[DISTANCE\_SENSORS\_NUMBER];

static const char \*distance\_sensors\_names[DISTANCE\_SENSORS\_NUMBER] = {"ps0", "ps1", "ps2", "ps3", "ps4", "ps5", "ps6", "ps7"};

#define GROUND\_SENSORS\_NUMBER 3

static WbDeviceTag ground\_sensors[GROUND\_SENSORS\_NUMBER];

static double ground\_sensors\_values[GROUND\_SENSORS\_NUMBER] = {0.0, 0.0, 0.0};

static const char \*ground\_sensors\_names[GROUND\_SENSORS\_NUMBER] = {"gs0", "gs1", "gs2"};

#define LEDS\_NUMBER 10

static WbDeviceTag leds[LEDS\_NUMBER];

static bool leds\_values[LEDS\_NUMBER];

static const char \*leds\_names[LEDS\_NUMBER] = {"led0", "led1", "led2", "led3", "led4", "led5", "led6", "led7", "led8", "led9"};

static WbDeviceTag left\_motor, right\_motor;

#define LEFT 0

#define RIGHT 1

#define MAX\_SPEED 6.28

static double speeds[2];

/\* Breitenberg stuff \*/

static double weights[DISTANCE\_SENSORS\_NUMBER][2] = {{-1.3, -1.0}, {-1.3, -1.0}, {-0.5, 0.5}, {0.0, 0.0},

{0.0, 0.0}, {0.05, -0.5}, {-0.75, 0}, {-0.75, 0}};

static double offsets[2] = {0.5 \* MAX\_SPEED, 0.5 \* MAX\_SPEED};

static int get\_time\_step() {

static int time\_step = -1;

if (time\_step == -1)

time\_step = (int)wb\_robot\_get\_basic\_time\_step();

return time\_step;

}

static void step() {

if (wb\_robot\_step(get\_time\_step()) == -1) {

wb\_robot\_cleanup();

exit(EXIT\_SUCCESS);

}

}

static void passive\_wait(double sec) {

double start\_time = wb\_robot\_get\_time();

do {

step();

} while (start\_time + sec > wb\_robot\_get\_time());

}

static void init\_devices() {

int i;

for (i = 0; i < DISTANCE\_SENSORS\_NUMBER; i++) {

distance\_sensors[i] = wb\_robot\_get\_device(distance\_sensors\_names[i]);

wb\_distance\_sensor\_enable(distance\_sensors[i], get\_time\_step());

}

for (i = 0; i < LEDS\_NUMBER; i++)

leds[i] = wb\_robot\_get\_device(leds\_names[i]);

// silently initialize the ground sensors if they exists

for (i = 0; i < GROUND\_SENSORS\_NUMBER; i++)

ground\_sensors[i] = (WbDeviceTag)0;

int ndevices = wb\_robot\_get\_number\_of\_devices();

for (i = 0; i < ndevices; i++) {

WbDeviceTag dtag = wb\_robot\_get\_device\_by\_index(i);

const char \*dname = wb\_device\_get\_name(dtag);

WbNodeType dtype = wb\_device\_get\_node\_type(dtag);

if (dtype == WB\_NODE\_DISTANCE\_SENSOR && strlen(dname) == 3 && dname[0] == 'g' && dname[1] == 's') {

int id = dname[2] - '0';

if (id >= 0 && id < GROUND\_SENSORS\_NUMBER) {

ground\_sensors[id] = wb\_robot\_get\_device(ground\_sensors\_names[id]);

wb\_distance\_sensor\_enable(ground\_sensors[id], get\_time\_step());

}

}

}

// get a handler to the motors and set target position to infinity (speed control).

left\_motor = wb\_robot\_get\_device("left wheel motor");

right\_motor = wb\_robot\_get\_device("right wheel motor");

wb\_motor\_set\_position(left\_motor, INFINITY);

wb\_motor\_set\_position(right\_motor, INFINITY);

wb\_motor\_set\_velocity(left\_motor, 0.0);

wb\_motor\_set\_velocity(right\_motor, 0.0);

step();

}

static void reset\_actuator\_values() {

int i;

for (i = 0; i < 2; i++)

speeds[i] = 0.0;

for (i = 0; i < LEDS\_NUMBER; i++)

leds\_values[i] = false;

}

static void get\_sensor\_input() {

int i;

for (i = 0; i < DISTANCE\_SENSORS\_NUMBER; i++) {

distance\_sensors\_values[i] = wb\_distance\_sensor\_get\_value(distance\_sensors[i]);

// scale the data in order to have a value between 0.0 and 1.0

// 1.0 representing something to avoid, 0.0 representing nothing to avoid

distance\_sensors\_values[i] /= 4096;

}

for (i = 0; i < GROUND\_SENSORS\_NUMBER; i++) {

if (ground\_sensors[i])

ground\_sensors\_values[i] = wb\_distance\_sensor\_get\_value(ground\_sensors[i]);

}

}

static bool cliff\_detected() {

int i;

for (i = 0; i < GROUND\_SENSORS\_NUMBER; i++) {

if (!ground\_sensors[i])

return false;

if (ground\_sensors\_values[i] < 500.0)

return true;

}

return false;

}

static void set\_actuators() {

int i;

for (i = 0; i < LEDS\_NUMBER; i++)

wb\_led\_set(leds[i], leds\_values[i]);

wb\_motor\_set\_velocity(left\_motor, speeds[LEFT]);

wb\_motor\_set\_velocity(right\_motor, speeds[RIGHT]);

}

static void blink\_leds() {

static int counter = 0;

counter++;

leds\_values[(counter / 10) % LEDS\_NUMBER] = true;

}

static void run\_braitenberg() {

int i, j;

for (i = 0; i < 2; i++) {

speeds[i] = 0.0;

for (j = 0; j < DISTANCE\_SENSORS\_NUMBER; j++)

speeds[i] += distance\_sensors\_values[j] \* weights[j][i];

speeds[i] = offsets[i] + speeds[i] \* MAX\_SPEED;

if (speeds[i] > MAX\_SPEED)

speeds[i] = MAX\_SPEED;

else if (speeds[i] < -MAX\_SPEED)

speeds[i] = -MAX\_SPEED;

}

}

static void go\_backwards() {

wb\_motor\_set\_velocity(left\_motor, -MAX\_SPEED);

wb\_motor\_set\_velocity(right\_motor, -MAX\_SPEED);

passive\_wait(0.2);

}

static void turn\_left() {

wb\_motor\_set\_velocity(left\_motor, -MAX\_SPEED);

wb\_motor\_set\_velocity(right\_motor, MAX\_SPEED);

passive\_wait(0.2);

}

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

wb\_robot\_init();

printf("Default controller of the e-puck robot started...\n");

init\_devices();

while (true) {

reset\_actuator\_values();

get\_sensor\_input();

blink\_leds();

if (cliff\_detected()) {

go\_backwards();

turn\_left();

} else {

run\_braitenberg();

}

set\_actuators();

step();

};

return EXIT\_SUCCESS;

}