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»

ОТЧЕТ

По лабораторной работе №2. Программирование алгоритмов управления роботом в Webots.

РУКОВОДИТЕЛЬ:

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

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Юрчук М.С.

(подпись) (фамилия, и.,о.)

19-В-1

(шифр группы)

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

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

Нижний Новгород 2022

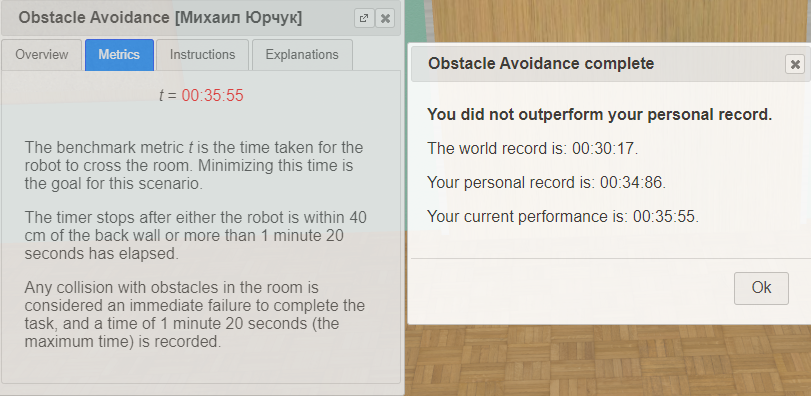
**Цель:** получение навыков работы с алгоритмами управления роботами.

**Задание**: выполнить вариант и загрузить программу на платформу для соревнований, записать видео работы, подготовить отчёт с подробным описанием результатов. В отчёт вставить результаты с соревнования. Есть примеры, можно на них посмотреть (видео на сайте).

**Задача 1. Обход препятствий**

Thymio II в базовом контроллере считывает значения с датчиков переднего расстояния и использует их для непосредственного управления скоростью своих колес. Чтобы успешно избегать препятствий с помощью механизма реагирования датчика, реализованного здесь, робот должен двигаться на полной скорости. Чтобы заставить робота двигаться на полной скорости, необходимо изменить силу воздействия датчиков расстояния на управление колесом.

Не зная абсолютного направления движения, робот может дезориентироваться и не достичь другой стороны комнаты. Для борьбы с этим Thymio II оснащен устройством компаса.



**Исходный код:**

"""Braitenberg-based obstacle-avoiding robot controller."""

from controller import Robot, Compass

def clamp(num, min\_value, max\_value):

return max(min(num, max\_value), min\_value)

# Get reference to the robot.

robot = Robot()

# get robot's Compass device

compass = robot.getCompass("compass")

# Get simulation step length.

timeStep = int(robot.getBasicTimeStep())

# Constants of the Thymio II motors and distance sensors.

maxMotorVelocity = 9.53

distanceSensorCalibrationConstant = 360

# enable the Compass

compass.enable(timeStep)

# Get left and right wheel motors.

leftMotor = robot.getMotor("motor.left")

rightMotor = robot.getMotor("motor.right")

# Get frontal distance sensors.

outerLeftSensor = robot.getDistanceSensor("prox.horizontal.0")

centralLeftSensor = robot.getDistanceSensor("prox.horizontal.1")

centralSensor = robot.getDistanceSensor("prox.horizontal.2")

centralRightSensor = robot.getDistanceSensor("prox.horizontal.3")

outerRightSensor = robot.getDistanceSensor("prox.horizontal.4")

# Enable distance sensors.

outerLeftSensor.enable(timeStep)

centralLeftSensor.enable(timeStep)

centralSensor.enable(timeStep)

centralRightSensor.enable(timeStep)

outerRightSensor.enable(timeStep)

# Disable motor PID control mode.

leftMotor.setPosition(float('inf'))

rightMotor.setPosition(float('inf'))

# Set ideal motor velocity.

initialVelocity = 0.7 \* maxMotorVelocity

# Set the initial velocity of the left and right wheel motors.

leftMotor.setVelocity(maxMotorVelocity)

rightMotor.setVelocity(maxMotorVelocity)

#mCentralSensorValue = []

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

# Read values from four distance sensors and calibrate.

outerLeftSensorValue = outerLeftSensor.getValue() / distanceSensorCalibrationConstant

centralLeftSensorValue = centralLeftSensor.getValue() / distanceSensorCalibrationConstant

centralSensorValue = centralSensor.getValue() / distanceSensorCalibrationConstant

centralRightSensorValue = centralRightSensor.getValue() / distanceSensorCalibrationConstant

outerRightSensorValue = outerRightSensor.getValue() / distanceSensorCalibrationConstant

#mCentralSensorValue.append(centralSensorValue)

#print(max(mCentralSensorValue))

# 9.045222213801566 - collison

maxValue = max(outerLeftSensorValue, centralLeftSensorValue, centralSensorValue, centralRightSensorValue, outerRightSensorValue)

#print("Max value: ", maxValue)

if centralSensorValue >= 0.01:

if outerLeftSensorValue > outerRightSensorValue:

leftMotor.setVelocity(maxMotorVelocity \* (-outerRightSensorValue / 100 \* 7.9))

rightMotor.setVelocity(maxMotorVelocity \* clamp((1.0 - maxValue / 10), 0.15, 1.0))

else:

leftMotor.setVelocity(maxMotorVelocity \* clamp((1.0 - maxValue / 10), 0.15, 1.0))

rightMotor.setVelocity(maxMotorVelocity \* (-outerLeftSensorValue / 100 \* 7.9))

elif (outerLeftSensorValue + centralLeftSensorValue) >= 0.01:

leftMotor.setVelocity(maxMotorVelocity \* clamp((1.0 - maxValue / 10), 0.15, 1.0))

rightMotor.setVelocity(maxMotorVelocity \* (-outerLeftSensorValue / 100 \* 7.9))

elif (centralRightSensorValue + outerRightSensorValue) >= 0.01:

leftMotor.setVelocity(maxMotorVelocity \* (-outerRightSensorValue / 100 \* 7.9))

rightMotor.setVelocity(maxMotorVelocity \* clamp((1.0 - maxValue / 10), 0.15, 1.0))

# to read values

values = compass.getValues()

if (outerLeftSensorValue + centralLeftSensorValue + centralSensorValue + centralRightSensorValue + outerRightSensorValue) <= 0.01:

#print(values)

if values[0] > 0.0001:

leftMotor.setVelocity(maxMotorVelocity)

rightMotor.setVelocity(maxMotorVelocity \* 0.777)

elif values[0] < -0.0001:

leftMotor.setVelocity(maxMotorVelocity \* 0.777)

rightMotor.setVelocity(maxMotorVelocity)

else:

leftMotor.setVelocity(maxMotorVelocity)

rightMotor.setVelocity(maxMotorVelocity)

**Задача 2**. Движение по квадрату.

Этот тест направлен на разработку программы, которая управляет роботом Pioneer, чтобы следовать квадратному пути размером 2 на 2 метра.

Метрика, используемая для оценки робота, применяется к 4 отдельным сегментам пути, которые соответствуют 4 сторонам квадрата.

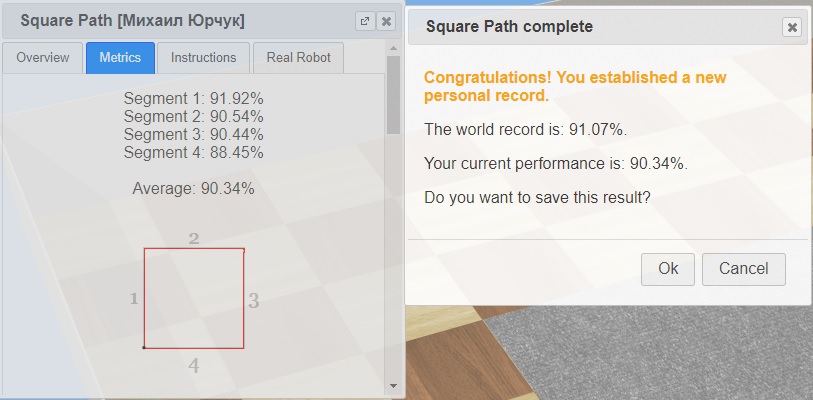
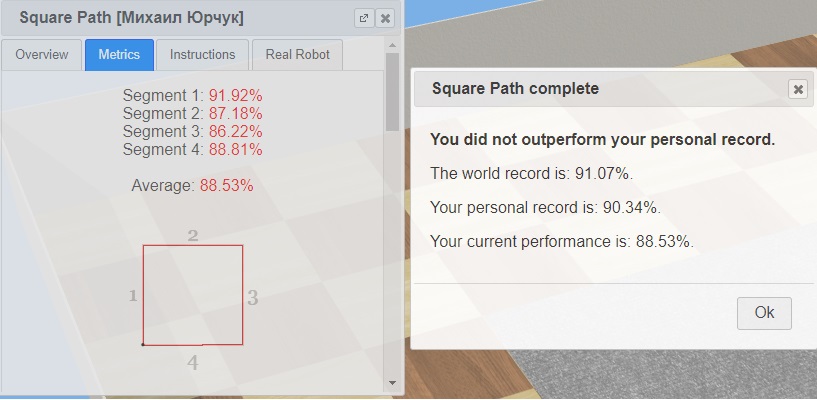
Каждый сегмент определяется как коридор, лежащий на одном краю квадрата. «Цель» одного сегмента определяется как вершина между текущим и следующим сегментом. Чтобы добраться до следующего сегмента, робот должен пересечь линию, проходящую через центр квадрата и вершину «цели».

Для каждого отдельного сегмента мы вычисляем производительность, которая основана на 3 различных параметрах: путь (насколько хорошо роботу удалось приблизиться к "идеальному" маршруту), время, необходимое для прохождения этого сегмента, и расстояние до цели, которое в основном используется для оценки того, насколько робот близок к цели в текущем сегменте.

Производительность для сегмента представляет собой средневзвешенное значение трех элементов. Элемент времени имеет половину веса двух других.

Общая производительность - это среднее значение по 4 сегментам.

Более грамотным решением было бы использовать датчики колес робота. Pioneer 3-DX имеет датчики в колесе, которые позволяют роботу знать пройденное расстояние.



**Исходный код:**

"""Sample Webots controller for the square path benchmark."""

from controller import Robot

import math

# Max velocity

MAX\_SPEED = 5.24

# Diameter in meteres

WHEEL\_DIAMETER = 0.195 # 195mm

# Distance between wheels in meteres

WHEEL\_DIST = 0.330 # 330mm

l = math.pi \* WHEEL\_DIAMETER

r = math.pi \* WHEEL\_DIST / 4.0 / l

c = 2.0 \* math.pi \* r

# Get pointer to the robot.

robot = Robot()

# Get pointer to each wheel of our robot.

leftWheel = robot.getDevice('left wheel')

rightWheel = robot.getDevice('right wheel')

# Get sensor for each wheel of our robot.

rightWheelSensor = robot.getDevice('right wheel sensor')

rightWheelSensor.enable(16) # Refreshes the sensor every 16ms.

leftWheelSensor = robot.getDevice('left wheel sensor')

leftWheelSensor.enable(16) # Refreshes the sensor every 16ms.

sensorValue = 0

# Repeat the following 4 times (once for each side).

for i in range(0, 4):

# First set both wheels to go forward, so the robot goes straight.

leftWheel.setPosition(1000)

rightWheel.setPosition(1000)

robot.step(16) # sensor updating every 16ms

# Moving untill the corner (2 meters)

while rightWheelSensor.getValue() \* WHEEL\_DIAMETER / 2.0 < sensorValue + 2.0:

if rightWheelSensor.getValue() \* WHEEL\_DIAMETER / 2.0 > sensorValue + 1.965:

leftWheel.setVelocity(MAX\_SPEED \* 0.60)

rightWheel.setVelocity(MAX\_SPEED \* 0.60)

robot.step(160)

leftError = 0.09

rightError = 0.05

if i == 1:

leftError = 0.07

rightError = 0.04

elif i == 2:

leftError = 0.088

rightError = 0.027

leftWheel.setPosition(leftWheelSensor.getValue() + c + leftError)

rightWheel.setPosition(rightWheelSensor.getValue() - c - rightError)

# Wait until the robot has turned 90 degrees clockwise.

robot.step(900)

sensorValue = rightWheelSensor.getValue() \* WHEEL\_DIAMETER / 2.0

leftWheel.setVelocity(MAX\_SPEED)

rightWheel.setVelocity(MAX\_SPEED)

# Stop the robot when path is completed, as the robot performance

# is only computed when the robot has stopped.

leftWheel.setVelocity(0)

rightWheel.setVelocity(0)

**Вывод**: мы написали несколько алгоритмов для робота. В первом случае для обхода препятствий, второй – движение по квадрату.