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

по дисциплине

«Аппаратное и программное обеспечение робототехнических систем»

РУКОВОДИТЕЛЬ:

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

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Соляник Д.Р.

ГРУППА:

19-В-1

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

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

Нижний Новгород

2022г.

Лабораторная работа № 2. Программирование алгоритмов управления роботом в Webots

**Цель:** получение навыков работы с симулятором Webots

**Задание:**

1. Обход препятствий

2. Движение по квадрату

**Ход работы:**

Обнаружение препятствия. На работе установлены сенсоры, чтобы заметить препятствий, мы используем их показания. Если препятствие расположено слева, то мы видим, что сенсоры, расположенные слева, меняют свое значение (относительно нуля). Следовательно, нам нужно повернуть направо. Замедлив правое колесо, совершаем поворот. Аналогично поступаем если препятствие находится справа.

В коде используем компас, чтобы при обходе препятствий робот выравнивался и ехал в нужном направлении, иначе при обходе он может потеряться в пространстве. Скорость выравнивания робота относительно финиша небольшая. Это дает нам гарантию, что робот обойдет препятствие, а не врежется к него.

Изменил контроллер:

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

# import Compass module

from controller import Compass

from controller import Robot

# Get reference to the robot.

robot = Robot()

# Get simulation step length.

timeStep = int(robot.getBasicTimeStep())

# Constants of the Thymio II motors and distance sensors.

maxMotorVelocity = 15

distanceSensorCalibrationConstant = 200

# Get left and right wheel motors.

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

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

# get robot's Compass device

compass = robot.getCompass("compass")

# enable the Compass

compass.enable(timeStep)

# 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(initialVelocity)

rightMotor.setVelocity(initialVelocity)

# flag = False

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

compassValues = compass.getValues()

print(compassValues)

leftMotor.setVelocity(initialVelocity)

rightMotor.setVelocity(initialVelocity)

# 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

compassLeft, \_, compassBack = compass.getValues()

if (outerLeftSensorValue != 0 or centralLeftSensorValue != 0 or centralSensorValue != 0 or centralRightSensorValue != 0 or outerRightSensorValue != 0):

# Set wheel velocities based on sensor values, prefer right turns if the central sensor is triggered.

leftMotor.setVelocity(initialVelocity - (centralRightSensorValue + outerRightSensorValue) - centralSensorValue)

rightMotor.setVelocity(initialVelocity - (centralLeftSensorValue + outerLeftSensorValue))

elif (compassBack > -0.9):

leftMotor.setVelocity(initialVelocity \* (1 + compassLeft/2))

rightMotor.setVelocity(initialVelocity \* (1 - compassLeft/2))

compassLeft, \_, compassBack = compass.getValues()

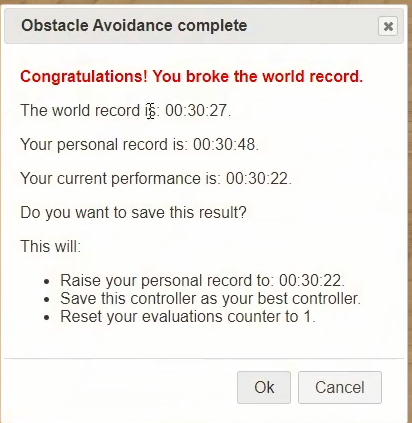
elif (compassBack > -0.999):

leftMotor.setVelocity(initialVelocity \* (1 + compassLeft))

rightMotor.setVelocity(initialVelocity \* (1 - compassLeft))

compassLeft, \_, compassBack = compass.getValues()

**Результат:**

****

**Ссылка:**

https://robotbenchmark.net/benchmark/obstacle\_avoidance/simulation.php?user=10512

**Ход работы:**

Провели несколько экспериментов и при помощи сенсора на правом колесе мы узнали длину прямой. Это значения мы приняли за константу, чтобы работать с ним на сегментах.

Проход первого сегмента самый простой. Нужно просто проехать по прямой.

После поворота на следующий сегмент происходит корректировка нашей константы. На повороте правое колесо немного отъезжает и значение константы изменяется в меньшую сторону.

Так как нам не удалось сделать чтобы робот поворачивал ровно на 90 градусов, получилась не стопроцентная точность. Углы, на которые поворачивал робот, были подобраны так, чтобы отклонение от ожидаемого маршрута было минимальным.

Изменил контроллер:

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

from controller import Robot

# Get pointer to the robot.

robot = Robot()

# Get pointer to each wheel of our robot.

leftWheel = robot.getMotor('left wheel')

rightWheel = robot.getMotor('right wheel')

#Get right wheel sensor

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

rightWheelSensor.enable(16)

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

for i in range(0, 4):

var = 0

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

leftWheel.setPosition(1000)

rightWheel.setPosition(1000)

print(rightWheelSensor.getValue()/(i+1))

if i == 0:

while round(rightWheelSensor.getValue(),1) != 20.6:

# Wait for the robot to reach a corner.

print(rightWheelSensor.getValue()/(i+1))

robot.step(var)

var += 1

if i == 1:

while round(rightWheelSensor.getValue()-(18\*i),1) < 20.8:

# Wait for the robot to reach a corner.

print(rightWheelSensor.getValue()/(i+1))

robot.step(var)

var += 1

if i == 2:

while round(rightWheelSensor.getValue()-(18\*i),1) < 21.1:

# Wait for the robot to reach a corner.

print(rightWheelSensor.getValue()/(i+1))

robot.step(var)

var += 1

if i == 3:

while round(rightWheelSensor.getValue()-(18\*i),1) < 20.9:

# Wait for the robot to reach a corner.

print(rightWheelSensor.getValue()/(i+1))

robot.step(var)

var += 1

# Then, set the right wheel backward, so the robot will turn right.

leftWheel.setPosition(1000)

rightWheel.setPosition(-1000)

# Wait until the robot has turned 90 degrees clockwise.

if i == 2 or i == 0:

robot.step(464)

else:

robot.step(465)

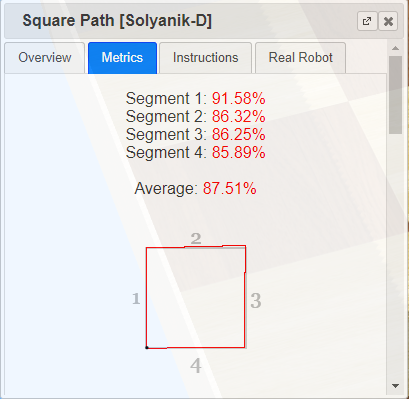
# Stop the robot when path is completed, as the robot performance

# is only computed when the robot has stopped.

leftWheel.setVelocity(0)

rightWheel.setVelocity(0)

**Результат:**

****

**Ссылка:**

https://robotbenchmark.net/benchmark/square\_path/simulation.php?user=10512