МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра вычислительных систем и технологий

«Программирование алгоритмов управления роботом в Webots»

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине

«Аппаратное и программное обеспечение роботизированных систем»

Вариант № 15

РУКОВОДИТЕЛЬ:

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

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

СТУДЕНТ:

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Марчус К.Р.

Гр. 19-В-1

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

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

Нижний Новгород 2022

# Цель работы

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).

Обход препятствий: <https://robotbenchmark.net/benchmark/obstacle_avoidance>

Движение по квадрату: <https://robotbenchmark.net/benchmark/square_path/>

# Алгоритм

**Обход препятствий**

Для того, чтобы робот видел препятствия, необходимо передавать показания сенсоров. Их мы записываем в массив. Чем больше показание сенсора, тем ближе препятствие. В зависимости от показателей сенсоров мы меняем скорость двигателей. Также предусмотрена возможность реверса робота (т.е. отката назад), если препятствия расположены со всех сторон.

Компас необходим для выравнивания траектории робота так, чтобы он отклонялся от препятствия по гиперболе (т.е.   
скорость двигателя / коэффициент) и со временем возвращался к начальному направлению движения и не врезался в препятствия, которые он обходил ранее.

**Движение по квадрату**

Алгоритм состоит из 4х сегментов. Начало каждого сегмента начинается с прохождения роботом прямой траектории. В конце сегмента робот поворачивает по часовой стрелке. После 4го сегмента робот полностью останавливается. Все значения, используемые в аргументах ф-ции step высчитаны эмпирическим методом.

# Код программы

**avoid\_obstacles.py:**

sensors\_dict = {

0: 'outer left',

1: 'front left',

2: 'front',

3: 'front right',

4: 'outer right',

}

# You may need to import some classes of the controller module. Ex:

# from controller import Robot, Motor, DistanceSensor

from controller import Robot, Compass

# create the Robot instance.

robot = Robot()

# get the time step of the current world.

timestep = int(robot.getBasicTimeStep())

# Constants of the Thymio II motors and distance sensors.

maxVelocity = 9.53

distanceSensorCalibrationConstant = 360

# get robot's Compass device

compass = robot.getCompass('compass')

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

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

sensors = [robot.getDistanceSensor('prox.horizontal.0'), robot.getDistanceSensor('prox.horizontal.1'),

robot.getDistanceSensor('prox.horizontal.2'), robot.getDistanceSensor('prox.horizontal.3'),

robot.getDistanceSensor('prox.horizontal.4')]

# Set ideal motor velocity.

currentVelocity = maxVelocity

for sensor in sensors:

sensor.enable(timestep)

compass.enable(timestep)

leftMotor.setPosition(float('inf'))

rightMotor.setPosition(float('inf'))

currentVelocity = maxVelocity

obstacle\_flag = False

obstacle\_count = 0

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

# Read the sensors:

compass\_values = compass.getValues()

sensor\_values = []

for sensor in sensors:

sensor\_values.append(sensor.getValue() / distanceSensorCalibrationConstant)

# Process sensor data here.

i = 0

print(sensor\_values)

if obstacle\_flag:

# if long obstacle

long\_obstacle\_flag = False

for value in sensor\_values[1:-1]:

if value != 0:

print('[Motor] facing long obstacle')

long\_obstacle\_flag = True

obstacle\_count = 4

break

if sensor\_values[0] > 8:

print('[Motor] facing long obstacle left')

long\_obstacle\_flag = True

obstacle\_count = 4

break

if sensor\_values[4] > 8:

print('[Motor] facing long right')

long\_obstacle\_flag = True

obstacle\_count = 4

break

if not long\_obstacle\_flag:

if obstacle\_count == 0:

obstacle\_flag = False

print('[Motor] facing no obstacle')

obstacle\_count -= 1

print(obstacle\_count)

# if rounded by obstacles

if sensor\_values[0] != 0 and sensor\_values[1] != 0 and sensor\_values[2] != 0 and sensor\_values[3] != 0 and sensor\_values[

4] != 0:

rightMotor.setVelocity(-currentVelocity)

leftMotor.setVelocity(-currentVelocity)

print('[Motor] reverse')

# if obstacle left

elif sensor\_values[0] != 0 or sensor\_values[1] != 0 or sensor\_values[2] != 0:

rightMotor.setVelocity(-currentVelocity)

leftMotor.setVelocity(currentVelocity)

print('[Motor] >> right')

# if obstacle right

elif sensor\_values[3] != 0 or sensor\_values[4] != 0:

rightMotor.setVelocity(currentVelocity)

leftMotor.setVelocity(-currentVelocity)

print('[Motor] << left')

# if no obstacle

else:

if sensor\_values[0] < 1 and sensor\_values[4] < 1 and sensor\_values[1] == 0 and sensor\_values[2] == 0 and sensor\_values[3] == 0:

if round(compass\_values[0], 3) < (-0.04) and sensor\_values[0] == 0:

rightMotor.setVelocity(currentVelocity)

leftMotor.setVelocity(currentVelocity / 1.17)

print('[Compass] stabilizing trajectory: << left')

elif round(compass\_values[0], 3) > 0.04 and sensor\_values[4] == 0:

rightMotor.setVelocity(currentVelocity / 1.17)

leftMotor.setVelocity(currentVelocity)

print('[Compass] stabilizing trajectory: >> right')

else:

rightMotor.setVelocity(currentVelocity)

leftMotor.setVelocity(currentVelocity)

else:

rightMotor.setVelocity(currentVelocity)

leftMotor.setVelocity(currentVelocity)

for value in sensor\_values[1:-1]:

if value != 0:

obstacle\_flag = True

obstacle\_count = 4

print('[Sensor] ', sensors\_dict[i], 'obstacle:')

i += 1

if sensor\_values[0] > 8:

obstacle\_flag = True

obstacle\_count = 4

print('[Sensor] ', sensors\_dict[i], 'obstacle:')

if sensor\_values[4] > 8:

obstacle\_flag = True

obstacle\_count = 4

print('[Sensor] ', sensors\_dict[i], 'obstacle:')

pass

**square\_path.py:**

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

def defolt():

leftWheel.setPosition(1000)

rightWheel.setPosition(1000)

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) # Refreshes the sensor every 16ms.

defolt()

# 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.

if i == 0:

robot.step(3900)

else:

while rightWheelSensor.getValue() / (i + 1) < 19:

robot.step(1)

print(rightWheelSensor.getValue())

# Wait for the robot to reach a corner.

#robot.step(3900)

# 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 == 0:

robot.step(464)

elif i == 1:

robot.step(464)

elif i == 2:

robot.step(465)

elif i == 3:

robot.step(465)

defolt()

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

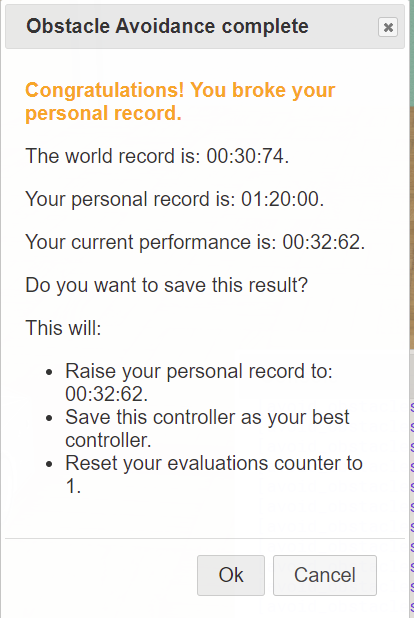
# is only computed when the robot has stopped.

leftWheel.setVelocity(0)

rightWheel.setVelocity(0)

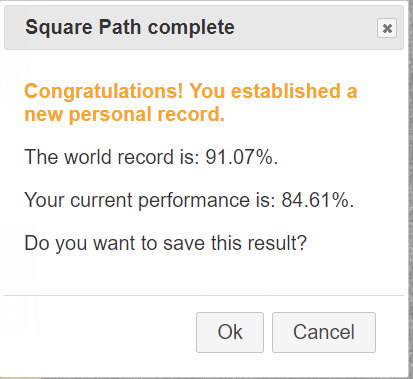
# Результат работы

Обход препятствий получилось пройти за 32.62 секунды (самый лучший результат). До этого были результаты в 35-40 секунд из-за неправильного написания алгоритма выравнивания траектории компасом (эти результаты не сохранялись).





Движение по квадрату, лучший результат 84.61%, в целом были результаты 80-83%





# Вывод

Мы получили базовые навыки работы с алгоритмами управления роботами.