МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального 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 роботизированных систем»

«Программирование алгоритмов управления роботов в Webots»

РУКОВОДИТЕЛЬ:

_____ Гай В.Е.

СТУДЕНТ:

_____ Ширшов А.А.

19-В-1

Работа защищена «___» _____

С оценкой _____

Нижний Новгород 2022

Цель: получение навыков работы с алгоримами управления роботами.

Задание: выполнить вариант и загрузить программу на платформу для соревнований, записать видео работы, подготовить отчет с подробным описанием результатов. В отчет вставить результаты с соревнования.

Задача 1. Обход препятствий.

Этот тест направлен на создание надежного и эффективного алгоритма обхода препятствий для робота Thymio II с использованием языка программирования Python. Цель состоит в том, чтобы робот пересек комнату и как можно быстрее достиг противоположной стены, избегая при этом всех столкновений с препятствиями.

Для проверки работоспособности алгоритма, препятствия располагаются случайным образом при каждом пробеге. Эталонная метрика t-это время, необходимое роботу, чтобы пересечь комнату. Минимизация этого времени является целью для этого сценария. Таймер останавливается после того, как робот находится в пределах 40 см от задней стенки или прошло более 1 минуты 20 секунд (максимальное время). Любое столкновение с препятствиями в комнате считается немедленным провалом задания.

Пояснение к коду:

Изначально контроллер робота был основан на транспортном средстве Брайтенберга (показания с датчиков управляют моторами робота). Это очень похоже на пропорциональный регулятор, поэтому возникла идея реализовать PID регулятор. Он получает значения с 5 датчиков расстояния. Получает среднее с двух левых и среднее с двух правых датчиков, если значение расстояния больше с левых датчиков, то к этому значению добавляется показание с центрального датчика, и наоборот, если у правых значение больше.

Затем эти 2 значения используются в пропорциональной, дифференциальной и интегральной составляющих. Касательно интегральной составляющей — используются 5 значений среднего расстояния с левых датчиков и 5 значений — с правых датчиков. При каждом вызове регулятора самое старое значение удаляется и добавляется новое. Таким образом:

error_left = up_left + ud_left + ui_left,
up_left = left * k_p
ud_left = (left — prev_left) * k_d
ui_left = sum(left[5]) * k_i

error_right = up_ right + ud_ right + ui_right,
up_ right = right * k_p
ud_right = (right — prev_right) * k_d
ui_ right = sum(right[5]) * k_i

Затем, если сумма ошибок маленькая, то есть вокруг нет препятствий, то робот ускоряется и выпрямляет себя по компасу по y = compass.getValues[1] (на сайте бенчмарка используется устаревшая версия, поэтому нужно значение в compass.getValues[0], кроме того, оно отрицательное). Если робот идет не по правильной траектории, то будет расти y, т.о чтобы робот ехал правильно, нужно из скорости конкретного колеса вычитать модуль y. Допустим робот съехал вправо, это значит, что y < 0 и уменьшается, поэтому нужно замедлить левое колесо.

Иначе если рядом препятствие, то мы должны замедлить противоположное колесо, т. е. ошибка слева большая (слева препятствие), значит из скорости правого колеса вычитаем эту ошибку, и наоборот.

Если ошибки слева и справа высоки (т. е. впереди стена препятствий), то робот отъезжает назад и снова пробует проехать.

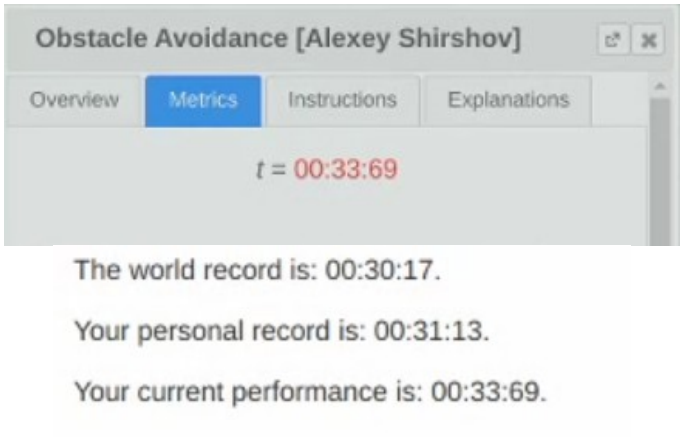
Исходный код:

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

from controller import Robot, Compass

# Get reference to the robot.
robot = Robot()

# Get simulation step length.
```

```
timeStep = int(robot.getBasicTimeStep())

# Constants of the Thymio II motors and distance sensors.
MAX_SPEED = 9.53
distanceSensorCalibrationConstant = 360
prev_y = 0

k_p = 0.001
k_d = 0.001
k_i = 0.003

sum_left = [0] * 5
sum_right = [0] * 5

max_ui = 6
min_ui = -6

# PID Controller for distance sensor
def PIDctl(y, sensors):
    global sum_left
    global sum_right

    # Get sensors values
    left = (sensors[0].getValue() + sensors[1].getValue()) /2
    right = (sensors[3].getValue() + sensors[4].getValue()) /2
    if (left > right):
        left += sensors[2].getValue()
    else:
        right += sensors[2].getValue()

    # Update values for integral
    sum_left.pop(0)
    sum_right.pop(0)
    sum_left.append(left)
    sum_right.append(right)

    # Integral
    ui_left = sum(sum_left) * k_i
    ui_right = sum(sum_right) * k_i

    if (ui_left > max_ui):
        ui_left = max_ui
    elif (ui_left < min_ui):
        ui_left = min_ui

    if (ui_right > max_ui):
        ui_right = max_ui
    elif (ui_right < min_ui):
        ui_right = min_ui

    # Proportional
    up_left = left * k_p
    up_right = right * k_p

    # Differential
    ud_left = (left - sum_left[3]) * k_d
    ud_right = (right - sum_right[3]) * k_d

    # Debug
    #print("%3.3f %3.3f %3.3f | %3.3f %3.3f %3.3f" % (up_left, ud_left, ui_left,
up_right, ud_right, ui_right))

    # return(up_left, up_right)
    return(up_left + ud_left + ui_left, up_right + ud_right + ui_right)

# Get left and right wheel motors.
leftMotor = robot.getMotor("motor.left")
rightMotor = robot.getMotor("motor.right")

# Get compass
compass = robot.getCompass("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 = 0.9 * MAX_SPEED

# Set the initial velocity of the left and right wheel motors.
leftMotor.setVelocity(initialVelocity)
rightMotor.setVelocity(initialVelocity)

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

    outerLeftSensorValue = outerLeftSensor.getValue() / distanceSensorCalibrationConstant
    centralLeftSensorValue = centralLeftSensor.getValue() /
distanceSensorCalibrationConstant
    centralSensorValue = centralSensor.getValue() / distanceSensorCalibrationConstant
    centralRightSensorValue = centralRightSensor.getValue() /
distanceSensorCalibrationConstant
    outerRightSensorValue = outerRightSensor.getValue() /
distanceSensorCalibrationConstant
    sensors = [outerLeftSensor, centralLeftSensor, centralSensor, centralRightSensor,
outerRightSensor]

    # Get errors
    robot.step(40)
    (error_left, error_right) = PIDctl(compass.getValues()[1], sensors)

    # If there is no obstacles, then need straight by compass
    if ((sum(sum_left) + sum(sum_right)) < 0.1):
        y = compass.getValues()[0]
        # if robot turned right
        if (y < -0.01):
            leftMotor.setVelocity(MAX_SPEED - abs(y)*5)
            rightMotor.setVelocity(MAX_SPEED)
        else:
            leftMotor.setVelocity(MAX_SPEED)
            rightMotor.setVelocity(MAX_SPEED - abs(y)*5)
    else:
        # Set wheel velocities based on sensor values.
        leftMotor.setVelocity(initialVelocity - error_right)
        rightMotor.setVelocity(initialVelocity - error_left)


    # If there is "wall"
    if (error_left > 6 and error_right > 6):
        # Then robot gets back
        leftMotor.setPosition(0)
        rightMotor.setPosition(0)
        leftMotor.setVelocity(0.1*MAX_SPEED)
        robot.step(1000)

    leftMotor.setPosition(float('inf'))
    rightMotor.setPosition(float('inf'))
```

Результат:


Obstacle Avoidance [Alexey Shirshov]
Overview
Metrics
Instructions
Explanations
t = 00:33:69
The world record is: 00:30:17.
Your personal record is: 00:31:13.
Your current performance is: 00:33:69.

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.

Пояснение к коду:

Для того, чтобы роботу точно пройти сторону квадрата необходимо ориентироваться на позиционные датчики, они возвращают пройденное расстояние в радианах, поэтому для удобства создана функция getDistance, которая по формуле L = R*a переводит радианы в метры и вычитает с прошлым значением, т.о. получаем пройденное расстояние на каждой стороне.

Когда робот проезжает почти 2 метра, он должен снизить скорость.

Затем ему нужно повернуться вокруг своей оси на четверть окружности (левое колесо крутится вперед, правое — назад). Определена переменная qurterInRads = (l*pi/4)/R, где l — расстояние между колесами, а R — радиус колеса. Она означает то, насколько должно колесо повернуться.

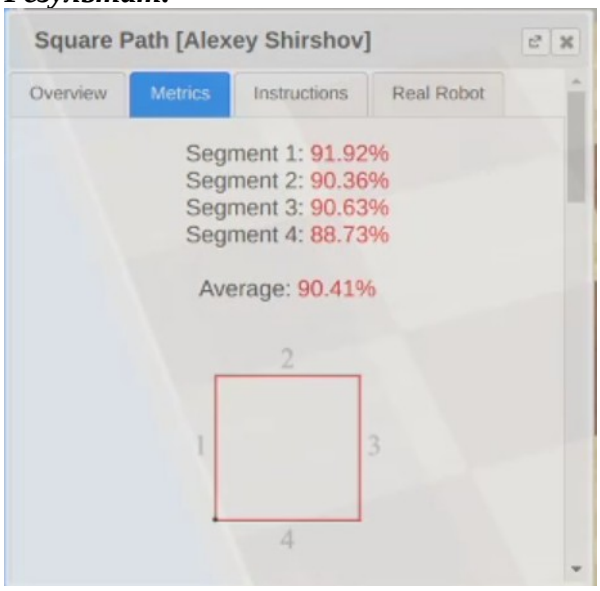
Из-за внешних факторов робот не полностью совершает поворот, поэтому необходимо еще добавить 0,5 рад — помогает во всех случаях, точность при этом 89%, однако, если нужна большая точность, то создан массив delta с подобранными значениями.

Исходный код:

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

from controller import Robot
import math

# Variables
wheelDiameter = 0.195
wheelRadius = wheelDiameter/2
wheelDistance = 0.33
MAX_SPEED = 5.24
prevValueSensor = 0

# delta = [[0.060, 0.055, 0.060, 0.04],
#          [ 0.065, 0.055, 0.060, 0.00]]

delta = [[0.065, 0.055, 0.060],
         [ 0.065, 0.045, 0.055]]

quarterInRads = (wheelDistance*math.pi/4)/wheelRadius

# Get distance from prev to current rads, returns meters
def getDistance(sensor, prevValue, radius=wheelRadius):
    return radius*sensor.getValue() - prevValue

# 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 sensors
leftWheelSensor = robot.getPositionSensor('left wheel sensor')
rightWheelSensor = robot.getPositionSensor('right wheel sensor')

# Enable sensors
leftWheelSensor.enable(16)
rightWheelSensor.enable(16)

leftWheel.setVelocity(MAX_SPEED)
rightWheel.setVelocity(MAX_SPEED)

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

    # While robot not reached corner
    while (getDistance(rightWheelSensor, prevValueSensor) < 2.0):
        # If corner is near then set slow speed
        if (getDistance(rightWheelSensor, prevValueSensor) > 1.95):
            leftWheel.setVelocity(0.6*MAX_SPEED)
            rightWheel.setVelocity(0.6*MAX_SPEED)
        robot.step(160)

    # robot spin
    if (side == 3):
        # Robot shouldn't spin at the last corner
        break
    else:
        leftWheel.setPosition(leftWheelSensor.getValue() + quarterInRads + delta[0]
[side])
        rightWheel.setPosition(rightWheelSensor.getValue() - quarterInRads - delta[1]
[side])

    robot.step(1000)

    # Update prev rads
    prevValueSensor = rightWheelSensor.getValue() * wheelRadius

    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)
```

Результат:



Вывод: в результате выполнения данной лабораторной работы были получены алгоритмы для управления роботов с помощью обратной связи (датчиков). Изучен принцип работы транспортного средства Брайтенберга. Проведены тесты реализованных программ на платформе для соревнований.