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»

ЛАБОРАТОРНАЯ РАБОТА №4

по дисциплине

Аппаратное и программное обеспечение роботизированных систем

РУКОВОДИТЕЛЬ:

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

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сипаков В.В.

(подпись) (фамилия, и.,о.)

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

(шифр группы)

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

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

Нижний Новгород 2022

# Цель работы

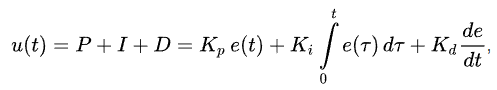
Получение навыков работы с алгоритмами управления роботами

# Задание

1. Перевернутый маятник (Inverted pendulum)

Этот тест направлен на разработку компьютерной программы, которая управляет колесным роботом, цель которого, как можно дольше удерживать маятник в равновесии. Язык программирования - Python, модель робота - робот e-puck. Во время моделирования к маятнику прикладывается некоторая случайная сила возмущения.

# Алгоритм



Используется ПИД-регулятор – функция, смешивающая 3 компонента:

* Пропорциональный - непосредственно разница между значением сенсора и желаемым результатом, умноженная на константу. Хорошо справляется в случаях, когда значения сильно отклоняются.
* Интегральный – поправка, учитывающая тенденции последних значений сенсоров. Смягчает результат, делая его более плавным, а также лучше контролирует значения около центральных, где пропорциональный компонент имеет мало эффекта.
* Дифференциальный – поправка, учитывающая текущую скорость изменения значений. Позволяет предотвратить колебания значений.

# Результат работы

Результат (1:00:32) получен путем изменения коэффициентов ПИД-регулятора. По умолчанию дифференциальная составляющая не используется, а интегральная немного завышена.

Экспериментальным образом найдены следующие конфигурации, стабильно выдающие результат ~1:00:20:

KP = 60.0, KI = 88.0, KD = 100.0

KP = 40.0, KI = 100.0, KD = 100.0

KP = 55.0, KI = 111.0, KD = 0.0

И др.

Обнаружены следующие закономерности:

* Изменение дифференциального коэффициента не сильно влияет на результат, при этом значение 100 всегда оказывается лучше нулевого значения.
* Изменение KI на значение <50 или >120 резко снижает результат.
* Изменение соотношения пропорционального коэффициента к интегральному влияет на чувствительность контроллера, чем оно выше, тем стабильней и резче контроллер реагирует на движение маятника. Чем оно ниже, тем больше осуществляется размашистых, колебательных движений.

**inverted\_pendulum.py:**

*"""Sample Webots controller for the inverted pendulum benchmark."""*from controller import Robot  
import math  
  
# Get pointer to the robot.  
robot = Robot()  
  
# Get the time step of the current world.  
timestep = int(robot.getBasicTimeStep())  
  
# Get pointers to the position sensor and enable it.  
ps = robot.getPositionSensor('pendulum sensor')  
ps.enable(timestep)  
  
# Get pointers to the motors and set target position to infinity (speed control).  
leftMotor = robot.getMotor("left wheel motor")  
rightMotor = robot.getMotor("right wheel motor")  
leftMotor.setPosition(float('+inf'))  
rightMotor.setPosition(float('+inf'))  
leftMotor.setVelocity(0.0)  
rightMotor.setVelocity(0.0)  
maxSpeed = min(rightMotor.getMaxVelocity(), leftMotor.getMaxVelocity())  
  
# Define the PID control constants and variables.  
KP = 60.0  
KI = 88.0  
KD = 100.0  
integral = 0.0  
previous\_position = 0.0  
  
# Initialize the robot speed (left wheel, right wheel).  
leftMotor.setVelocity(0.0)  
rightMotor.setVelocity(0.0)  
  
# Main loop: perform a simulation step until the simulation is over.  
while robot.step(timestep) != -1:  
 # Read the sensor measurement.  
 position = ps.getValue()  
  
 # Stop the robot when the pendulum falls.  
 if math.fabs(position) > math.pi \* 0.5:  
 leftMotor.setVelocity(0.0)  
 rightMotor.setVelocity(0.0)  
 break  
  
 # PID control.  
 integral = integral + (position + previous\_position) \* 0.5 / timestep  
 derivative = (position - previous\_position) / timestep  
 speed = KP \* position + KI \* integral + KD \* derivative  
  
 # Clamp speed to the maximum speed.  
 if speed > maxSpeed:  
 speed = maxSpeed  
 elif speed < -maxSpeed:  
 speed = -maxSpeed  
  
 # Set the robot speed (left wheel, right wheel).  
 leftMotor.setVelocity(-speed)  
 rightMotor.setVelocity(-speed)  
  
 # Store previous position for the next controller step.  
 previous\_position = position