ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО

ОБРАЗОВАНИЯ

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ИМ. Р.Е. АЛЕКСЕЕВА

**ЛАБОРАТОРНАЯ РАБОТА №4**

**Выполнил**: Тронин А.А. 19-В-1

**Проверил**: Гай В.Е.

**Оценка**: \_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород

2021

**Тема**: **Программирование алгоритмов управления роботом в Webots**

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

**Задание:** **Вариант 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.

Эталонная метрика t - это время, прошедшее с начала моделирования, причем наибольшее значение является лучшим. Измерение времени прекратится, как только маятник упадет вниз. Значение F отображает последнюю величину силы возмущения в ньютонах, которая была приложена к маятнику. Это значение силы будет линейно увеличиваться со временем.

**Как повысить устойчивость маятника?**

Пример программы на Python, управляющей роботом e-puck, использует ПИД-контроллер для установки скорости колес:

 # PID control

  integral = integral + (position + previous\_position) \* 0.5 / timestep

  derivative = (position - previous\_position) / timestep

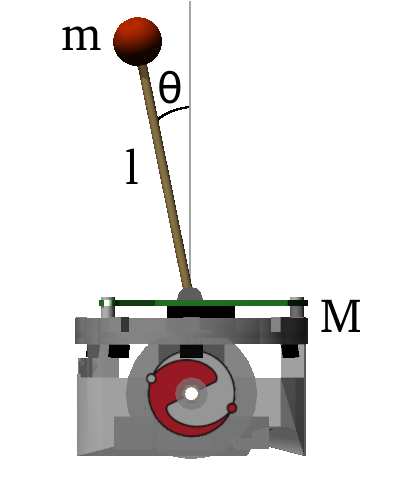
  speed = KP \* position + KI \* integral + KD \* derivative

Значения коэффициентов ПИД-контроллера KP, KI и KD могут быть настроены для улучшения стабильности маятника и достижения лучшего результата.

В качестве альтернативы, можно улучшить результат, используя более сложный алгоритм, например, основанный на полученных уравнениях движения перевернутого маятника.

Конфигурация системы:

* масса маятника: m = 50 г
* длина маятника: l = 0.08 м
* масса робота e-puck: M = 160 г



**Код:**

"""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 = 500

KI = KP\*0.5

KD = KP\*0.25

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

