ДОДАТОК А

Лістинг програмного забезпечення контролера робота в середовищі Webots

```
from controller import Robot, DistanceSensor, Motor
import math
import time

class EPuckMultiCurveNavigation:
    def __init__(self):
        self.robot = Robot()
        self.timestep = int(self.robot.getBasicTimeStep())

        self.ps_names = ['ps0', 'ps1', 'ps2', 'ps3', 'ps4', 'ps5', 'ps6', 'ps7']
        self.distance_sensors = []
        for name in self.ps_names:
            sensor = self.robot.getDevice(name)
            sensor.enable(self.timestep)
            self.distance_sensors.append(sensor)

        self.left_motor = self.robot.getDevice('left wheel motor')
        self.right_motor = self.robot.getDevice('right wheel motor')
        self.left_motor.setPosition(float('inf'))
        self.right_motor.setPosition(float('inf'))

        self.wheel_radius = 0.0205
        self.axle_length = 0.052
        self.max_speed = 6.28

        self.path_points = []
        self.current_path_index = 0
        self.current_curve_index = 0
        self.estimated_pos = [0.0, 0.0]
        self.estimated_angle = 0.0
        self.last_time = self.robot.getTime()

        self.safety_distance = 400
        self.critical_distance = 200

        self.curve_sequence = self.define_curve_sequence()
        self.enforce_c1_continuity()

        self.goal_point = self.curve_sequence[-1]['control_points'][-1]

        self.has_replanned = False

        print("Multi-Curve Navigation initialized")
```

```
def define_curve_sequence(self):
    return [
        {
            'name': 'Крива 1',
            'control_points': [
                [0.0, 0.0],
                [0.0, 0.2],
                [0.1, 0.4],
                [0.2, 0.5]
            ],
            'points_count': 40
        },
        {
            'name': 'Крива 2',
            'control_points': [
                [0.2, 0.5],
                [0.3, 0.8],
                [0.3, 0.9],
                [0.25, 1.2]
            ],
            'points_count': 50
        },
        {
            'name': 'Крива 3',
            'control_points': [
                [0.25, 1.2],
                [0.3, 1.5],
                [-0.5, 1.8],
                [-0.4, 2.0]
            ],
            'points_count': 45
        },
        {
            'name': 'Крива 4',
            'control_points': [
                [-0.4, 2.0],
                [-0.4, 2.6],
                [0.0, 2.6],
                [0.0, 2.8]
            ],
            'points_count': 45
        },
        {
            'name': 'Крива 5',
            'control_points': [
                [0.0, 2.8],
                [-0.3, 2.5],
                [0.0, 3.4],
```

```
                [-0.5, 3.4]
            ],
            'points_count': 45
        }
    ]

    def enforce_c1_continuity(self):

        for i in range(1, len(self.curve_sequence)):
            prev_cp = self.curve_sequence[i - 1]['control_points']
            cp = self.curve_sequence[i]['control_points']

            cp[0] = prev_cp[3][:]

            p0 = cp[0]
            p2_prev = prev_cp[2]
            cp[1] = [
                p0[0] + (p0[0] - p2_prev[0]),
                p0[1] + (p0[1] - p2_prev[1])
            ]

            self.curve_sequence[i]['control_points'] = cp

    def cubic_bezier(self, p0, p1, p2, p3, t):

        x = (1 - t) ** 3 * p0[0] + 3 * (1 - t) ** 2 * t * p1[0] + 3 * (1 - t) * t ** 2 * p2[0] + t ** 3 *
p3[0]
        y = (1 - t) ** 3 * p0[1] + 3 * (1 - t) ** 2 * t * p1[1] + 3 * (1 - t) * t ** 2 * p2[1] + t ** 3 *
p3[1]
        return [x, y]

    def generate_bezier_path(self, control_points, num_points=50):

        path = []
        for i in range(num_points):
            t = i / (num_points - 1)
            point = self.cubic_bezier(
                control_points[0], control_points[1],
                control_points[2], control_points[3], t
            )
            path.append(point)
        return path

    def load_next_curve(self):

        if self.current_curve_index < len(self.curve_sequence):
            curve = self.curve_sequence[self.current_curve_index]
            print(f"\n {curve['name']}")
```

```
            print(f"Точки: {curve['control_points']}")

            self.path_points = self.generate_bezier_path(
                curve['control_points'],
                curve['points_count']
            )
            self.current_path_index = 0
            self.current_curve_index += 1
            return True
        else:
            print("Остання крива в масиві завершена!")
            return False

    def get_sensor_data(self):
        sensor_values = [sensor.getValue() for sensor in self.distance_sensors]
        return {
            'front_right': sensor_values[0],
            'right_front': sensor_values[1],
            'right':       sensor_values[2],
            'right_back':   sensor_values[3],
            'back_left':    sensor_values[4],
            'left_back':    sensor_values[5],
            'left':        sensor_values[6],
            'left_front':   sensor_values[7],
            'front': max(sensor_values[0], sensor_values[7]),
            'all_sensors': sensor_values
        }

    def check_obstacles(self, sensor_data):

        if (sensor_data['front_right'] > self.critical_distance or
                sensor_data['left_front'] > self.critical_distance):
            return "EMERGENCY_STOP"

        if (sensor_data['front'] > self.safety_distance or
                sensor_data['front_right'] > self.safety_distance or
                sensor_data['left_front'] > self.safety_distance):

            left_side = min(sensor_data['left'], sensor_data['left_front'])
            right_side = min(sensor_data['right'], sensor_data['front_right'])

            if left_side < right_side:
                return "AVOID_RIGHT"
            else:
                return "AVOID_LEFT"

        return "CLEAR"
```

```
def calculate_steering(self, target_point, dt):

    dx = target_point[0] - self.estimated_pos[0]
    dy = target_point[1] - self.estimated_pos[1]
    target_angle = math.atan2(dy, dx)

    angle_error = target_angle - self.estimated_angle
    while angle_error > math.pi:
        angle_error -= 2 * math.pi
    while angle_error < -math.pi:
        angle_error += 2 * math.pi

    distance = math.sqrt(dx * dx + dy * dy)

    base_speed = min(self.max_speed * 0.5, distance * 2.0)
    steering_gain = 2.0

    left_speed = base_speed - steering_gain * angle_error
    right_speed = base_speed + steering_gain * angle_error

    left_speed = max(min(left_speed, self.max_speed), -self.max_speed)
    right_speed = max(min(right_speed, self.max_speed), -self.max_speed)

    v_left = left_speed * self.wheel_radius
    v_right = right_speed * self.wheel_radius
    linear_velocity = (v_left + v_right) / 2.0
    angular_velocity = (v_right - v_left) / self.axle_length

    self.estimated_angle += angular_velocity * dt
    self.estimated_pos[0] += linear_velocity * math.cos(self.estimated_angle) * dt
    self.estimated_pos[1] += linear_velocity * math.sin(self.estimated_angle) * dt

    return left_speed, right_speed, distance

def get_lookahead_point(self):
    if not self.path_points or self.current_path_index >= len(self.path_points):
        return None

    for i in range(self.current_path_index,
                   min(self.current_path_index + 8, len(self.path_points))):
        point = self.path_points[i]
        dx = point[0] - self.estimated_pos[0]
        dy = point[1] - self.estimated_pos[1]
        distance = math.sqrt(dx * dx + dy * dy)

        if distance >= 0.15:
            return point
```

```
    return self.path_points[min(self.current_path_index, len(self.path_points) - 1)]

def emergency_backoff(self, turn_direction, duration=0.7, speed_ratio=0.3):

    print(f"EMERGENCY_STOP: Від'їзд назад з напрямком руху {turn_direction}...")

    base = self.max_speed * speed_ratio
    start_time = self.robot.getTime()

    while self.robot.step(self.timestep) != -1:
        current_time = self.robot.getTime()
        dt = current_time - self.last_time
        self.last_time = current_time

        if turn_direction == "LEFT":
            left_speed = -base * 0.4
            right_speed = -base
        elif turn_direction == "RIGHT":
            left_speed = -base
            right_speed = -base * 0.4
        else:
            left_speed = -base
            right_speed = -base

        self.left_motor.setVelocity(left_speed)
        self.right_motor.setVelocity(right_speed)

        v_left = left_speed * self.wheel_radius
        v_right = right_speed * self.wheel_radius
        linear_velocity = (v_left + v_right) / 2.0
        angular_velocity = (v_right - v_left) / self.axle_length

        self.estimated_angle += angular_velocity * dt
        self.estimated_pos[0] += linear_velocity * math.cos(self.estimated_angle) * dt
        self.estimated_pos[1] += linear_velocity * math.sin(self.estimated_angle) * dt

        if current_time - start_time >= duration:
            break

    self.left_motor.setVelocity(0.0)
    self.right_motor.setVelocity(0.0)
    print("Від'їзд назад завершено")

def replan_bezier_to_goal(self, avoidance_direction):


    p0 = self.estimated_pos[:]
    p3 = self.goal_point[:]
```

```
        heading = [math.cos(self.estimated_angle),
                   math.sin(self.estimated_angle)]

        if avoidance_direction == "AVOID_RIGHT":
            side = [-heading[1], heading[0]]
        else:
            side = [heading[1], -heading[0]]

        k1_forward = 0.25
        k1_side = 0.20
        k2_side = 0.30

        p1 = [
            p0[0] + heading[0] * k1_forward + side[0] * k1_side,
            p0[1] + heading[1] * k1_forward + side[1] * k1_side
        ]

        mid = [(p0[0] + p3[0]) / 2.0,
               (p0[1] + p3[1]) / 2.0]

        p2 = [
            mid[0] + side[0] * k2_side,
            mid[1] + side[1] * k2_side
        ]

        control_points = [p0, p1, p2, p3]

        for i, p in enumerate(control_points):
            print(f"  P{i}: {p}")

        self.path_points = self.generate_bezier_path(control_points, num_points=80)
        self.current_path_index = 0
        self.current_curve_index = len(self.curve_sequence)
        self.has_replanned = True

    def run_navigation(self):

        if not self.load_next_curve():
            return

        step_count = 0

        while self.robot.step(self.timestep) != -1:
            current_time = self.robot.getTime()
            dt = current_time - self.last_time
            self.last_time = current_time
```

```
            sensor_data = self.get_sensor_data()
            obstacle_status = self.check_obstacles(sensor_data)

            if obstacle_status == "EMERGENCY_STOP":
                print(f"Екстренна зупинка: frontR={sensor_data['front_right']:.0f}, "
                      f"frontL={sensor_data['left_front']:.0f}")

                if sensor_data['front_right'] > sensor_data['left_front']:
                    turn_dir = "LEFT"
                else:
                    turn_dir = "RIGHT"

                self.emergency_backoff(turn_dir, duration=5.0, speed_ratio=0.4)
                continue


            if obstacle_status in ["AVOID_LEFT", "AVOID_RIGHT"] and not self.has_replanned:
                print(f"Перешкода ({obstacle_status})")
                self.replan_bezier_to_goal(obstacle_status)

            target_point = self.get_lookahead_point()
            if target_point is None:
                print(f"Крива {self.current_curve_index - 1} завершена!")
                if self.has_replanned or not self.load_next_curve():
                    print("Кінець навігації")
                    self.left_motor.setVelocity(0)
                    self.right_motor.setVelocity(0)
                    break
                continue
            left_speed, right_speed, distance = self.calculate_steering(target_point, dt)
            if distance < 0.08:
                self.current_path_index += 1
            self.left_motor.setVelocity(left_speed)
            self.right_motor.setVelocity(right_speed)
            step_count += 1
            if step_count % 150 == 0 and self.path_points:
                curve_progress = (self.current_path_index / len(self.path_points)) * 100
                print(f"Прогресс кривої: {curve_progress:.1f}% | ")
if __name__ == "__main__":
    controller = EPuckMultiCurveNavigation()
    controller.run_navigation()
```

ДОДАТОК Б
Демонстраційний матеріал

Формат	Позиція.	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кільк.	Примітки
			Текстові документи		
А4	1	ГЮИК 442351.002 ПЗ			
			Пояснювальна записка	70 с.	
			Інші документи		
	2		Лістинг програми	2 с.	
	3		Демонстраційний матеріал	15 с.	

Змін.	Арк	Номер докум.	Підпис	Дата	ГЮИК.442351.002 ВД		
Розроб.		Мацько Д. В.			Підвищення точності маршруту мобільних роботів за Допомогою адаптованих кривих Безьє Відомість кваліфікаційної роботи	Літера	Аркуш	Аркуш
Перевір.		Демська Н. П.				Н	1	1
						Кафедра КІТАР ХНУРЕ		
Н.контр.		Теслюк С.І.						
Затв.		Невлюдов І. Ш.						