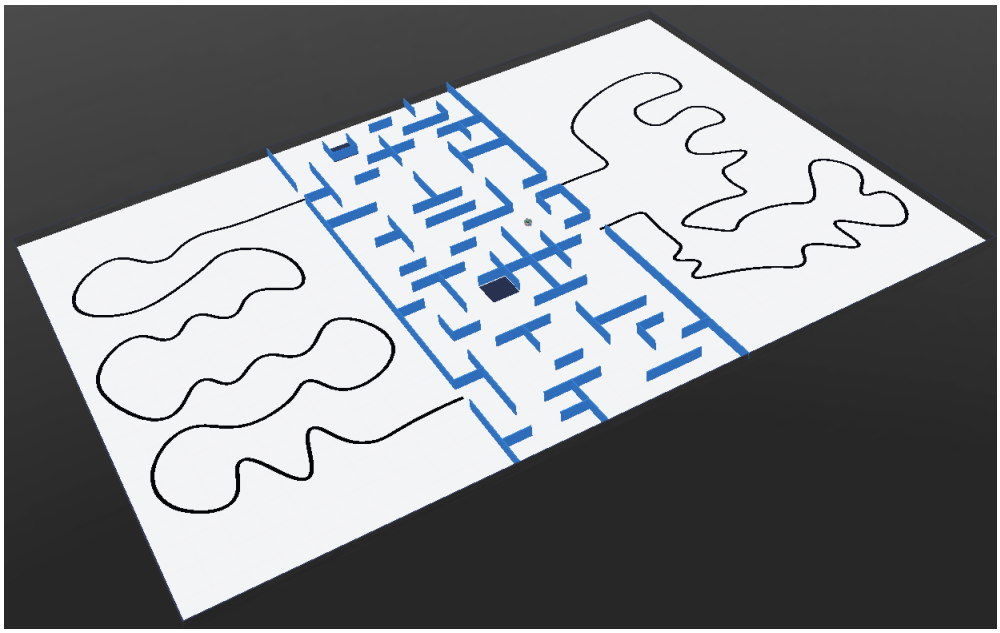
Computer Controlled System Lab: Mini Project 2

Due on July 5, 2024 at 11:59pm

Amirkabir University of Technology

Setayeshi, Hamidi

This mini-project encompasses two main tasks: a line-following task and a maze-solving task. The line-following task has been covered extensively as a class example during the semester, and the maze-solving task was the topic of the first mini-project. These foundational elements make this mini-project manageable and straightforward.



As you work through the assignments, please consider the following remarks:

1. Complete all exercises in Webots 2023b.
2. You are allowed (and it's even recommended!) to use AI tools like ChatGPT to assist you with your learning, but remember that mindless copying defeats the purpose. Interact actively, ask questions, understand the code, and explain it briefly in your reports. You are responsible for the content you submit, so remember that you should not blindly accept the written code from AI, even if it produces some outputs without error!

1 Project Objective

The general task of this mini-project is to start from a designated starting point in the maze, then navigate through the maze to locate two line-following zones. These zones contain a black line on a white background that the robot must follow. The robot should enter each line-following zone through one of its inputs, complete the line-following task, and then return to the maze. The specific entry and exit points for the line-following zones are unknown, but the dimensions of the maze and the positions of colored squares within the maze are provided.

2 Project Requirements

1. Maze Navigation and Line Following
   - **Starting Point:** Begin at the designated starting point in the maze.
   - **Maze Navigation:** Use appropriate algorithms to explore the maze and find the two line-following zones.

- **Line-Following Zones:** Upon entering a line-following zone, the robot must follow the black line until the task is completed, then re-enter the maze.
- **Completion:** After completing the line-following tasks in both zones, return to the starting point.

2. Control System Implementation

- **Write a PID Class:** Write a PID class using an appropriate discretization method.
- **PID Controller:** Implement a PID controller for various elementary tasks (e.g., moving straight, turning left, turning right). Avoid open-loop control approaches for tasks that should be modeled as control problems.
- **Task Integration:** Utilize the PID controller for the mentioned tasks, ensuring the robot performs them efficiently and accurately. In addition, do not forget to model the line-following task as a control problem, similar to the approach demonstrated in class.

3 Implementation Guidelines

1. PID Controller Design

- Develop a PID controller using a proper discretization method (e.g., Euler, Tustin).
- Implement the PID controller in a class structure, allowing for easy integration with different tasks.

2. Maze Exploration

- Utilize maze-solving algorithms to navigate and explore the maze.
- Incorporate strategies to find the line-following zones efficiently.

3. Line-Following Algorithm

- Implement a line-following algorithm using the PID controller to maintain the robot's trajectory along the black line.
- Ensure the robot can adapt to curves within the line-following zones.

4. Return to Starting Point

- After completing both line-following tasks, navigate back to the starting point in the maze.
- Implement a mechanism to record the robot's path, enabling it to retrace its steps if necessary. Do not forget that the location of the colored squares is known!

4 Base Code

4.1 Overview

We've provided you with a base code that serves as a starting point for your maze solving and line following project. Let's break down what's included:

1. **Reading Sensors:** The code includes functionalities to read from various sensors on the e-puck robot:

- Ground Sensors (gs): These sensors detect lines on the ground, helping the robot to follow lines or detect intersections.

```
# Ground sensors
gs = []
gsNames = ['gs0', 'gs1', 'gs2']
for i in range(3):
    gs.append(robot.getDevice(gsNames[i]))
    gs[i].enable(timestep)
```

- Proximity Sensors (ps): These sensors detect obstacles around the robot, aiding in obstacle avoidance.

```
# Proximity sensors
ps = []
psNames = ['ps0', 'ps1', 'ps2', 'ps3', 'ps4', 'ps5', 'ps6', 'ps7']
for i in range(8):
    ps.append(robot.getDevice(psNames[i]))
    ps[i].enable(timestep)
```

- Encoders (encoder): These sensors measure the rotation of the wheels, which can be used for precise distance and turning control.

```
# Encoders
encoder = []
encoderNames = ['left wheel sensor', 'right wheel sensor']
for i in range(2):
    encoder.append(robot.getDevice(encoderNames[i]))
    encoder[i].enable(timestep)

oldEncoderValues = []  # Initialize list to store previous encoder values
```

2. **Actuators Control:** You can control the e-puck robot's movement.

- These are used to set the speed and direction of the wheels, enabling the robot to move forward, backward, turn left, or turn right:

```
leftMotor = robot.getDevice('left wheel motor')
rightMotor = robot.getDevice('right wheel motor')

# Set motor position to infinity (unlimited rotation)
leftMotor.setPosition(float('inf'))
rightMotor.setPosition(float('inf'))

# Set initial motor velocities to 0
leftMotor.setVelocity(0.0)
rightMotor.setVelocity(0.0)
```

3. Obstacle Avoidance Logic: The base code includes a simple obstacle avoidance logic:

```
    # Adjust speeds based on obstacle detection
    if left_obstacle:
        # Turn right
        leftSpeed = 0.5 * MAX_SPEED
        rightSpeed = -0.5 * MAX_SPEED
    elif right_obstacle:
        # Turn left
        leftSpeed = -0.5 * MAX_SPEED
        rightSpeed = 0.5 * MAX_SPEED

    # Set motor velocities
    leftMotor.setVelocity(leftSpeed)
    rightMotor.setVelocity(rightSpeed)
```

4.2 Feedback Mechanisms

1. **Encoders:** Use encoder feedback to implement precise distance control for moving straight or making accurate turns. You don't need to use advanced functions (that I used in the video lecture) for robot navigation initially.

```
    encoderValues = [encoder[i].getValue() for i in range(2)]  # Read encoder values

    # Update old encoder values if not done before
    if len(oldEncoderValues) < 2:
        oldEncoderValues = encoderValues[:]
```

2. **Distance Sensors:** Integrate distance sensors to improve obstacle detection or to measure distances to maze walls for navigation tasks.

```
    # Detect obstacles from proximity sensors
    right_obstacle = any(psValues[i] > 80.0 for i in [0, 1, 2])  # Right proximity
    sensors
    left_obstacle = any(psValues[i] > 80.0 for i in [5, 6, 7])  # Left proximity
    sensors
```

5 Deliverables

1. Code Implementation
   - Submit the complete code for the maze-solving and line-following tasks.
   - Include the PID controller class and its integration with the tasks.
2. Documentation
   - Provide a detailed explanation of the PID controller design and implementation. Describe the maze-solving and line-following algorithms used.
   - Include a flowchart or diagram illustrating the overall process and control flow.
3. Demonstration
   - Present a video demonstration of the robot completing the mini-project tasks.
   - Ensure the video clearly shows the robot starting from the starting point, navigating the maze, completing the line-following tasks, and returning to the starting point.

6 Evaluation Criteria

1. Functionality
   - Correct implementation of the maze-solving and line-following tasks.
   - Successful navigation and completion of both line-following zones.
   - Accurate return to the starting point.
2. Control System
   - Effective use of the PID controller for various tasks.
   - Proper tuning the controller parameters and avoidance of open-loop approaches.

3. Code Quality
    - Clean, well-documented, and maintainable code.
    - Appropriate use of classes, functions, and modular design.
4. Presentation
    - Clear and concise documentation.
    - Comprehensive demonstration video showing all required tasks.

7 Homework Guidelines and Instructions

- The deadline for sending this exercise will be until the end of Friday, July 5.
- This exercise is done by a group of students.
- If any similarity is observed in the work report or implementation codes, this will be considered fraud for the parties.
- If you do not follow the format of the work report, you will not be awarded the grade of the report.
- All pictures and tables used in the work report must have captions and numbers.
- A large part of your grade is related to the work report and problem-solving process.
- Be happy and healthy!

Make it simple, make it work!
Setayeshi, Hamidi.