Computer Controlled System Lab: Mini Project 1

Due on April 26, 2024 at 11:59pm

Amirkabir University of Technology

Hamidi, Setayeshi

This homework is all about getting warmed up for coding in Python, as it will be a key asset for the rest of the course. So, let's put our "coding boots" on! This homework aims to review some simple and key concepts of programming. 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 primary objective of this project is to develop and demonstrate proficiency in implementing maze traversal algorithms using a simulated robotic environment in Webots 2023b. Students are expected to program an e-puck robot to autonomously navigate and traverse the entirety of a given maze.
The project will focus on two fundamental algorithms:

1. **Depth-First Search (DFS)**: This algorithm explores as far as possible along each branch before backtracking. It is well-suited for scenarios where you need to explore in a direction as long as possible and then backtrack to explore other paths.
2. **Breadth-First Search (BFS)**: This algorithm explores all the nearest nodes at the present depth prior to moving on to nodes at the next depth level. It is optimal for finding the shortest path and ensuring that all accessible areas of the maze are explored.

Students will select one of these algorithms to implement. The chosen algorithm should allow the e-puck robot to efficiently navigate through the maze, reaching all possible areas and demonstrating the ability to return to the start point of the maze. The success of the project will be measured by the robot's ability to:

- Successfully navigates and maps the entire maze.
- Demonstrate the chosen traversal algorithm's effectiveness in real-time simulation.
- Execute the traversal with minimal errors and without manual intervention.
- Returns to the start point of the maze.

This project not only tests technical skills in algorithm implementation but also assesses problem-solving capabilities and understanding of autonomous robotic navigation principles. It is an excellent opportunity for students to apply theoretical knowledge in a practical, engaging environment.
Students are required to document their process, challenges faced, and solutions implemented throughout the project, contributing to a final report that discusses their approach, results, and learnings.

2 Homework Guidelines and Instructions

- The deadline for sending this exercise will be until the end of Friday, April 26.
- This exercise is done by one person.
- 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.
- The following attachments are provided to assist you in completing this mini-project. Please review them, and if you have any questions or concerns, feel free to contact the teaching assistants via email.
- Be happy and healthy!

A Installation Guide for Webots 2023b on Ubuntu Linux Systems

A.1 Introduction

This guide provides detailed instructions on installing Webots 2023b on Ubuntu Linux systems. Webots is a widely used simulation software that supports various robotics projects. For Ubuntu, the installation can be done using different methods, depending on your preferences and system setup.

A.2 Prerequisites

- Ubuntu system (the instructions are tailored for recent distributions)
- Internet connection
- Sudo privileges on your system

A.3 Installation Methods

A.3.1 Method 1: Installing the Debian Package with APT

1. **Setup the Cyberbotics Repository:**

```
sudo mkdir -p /etc/apt/keyrings
cd /etc/apt/keyrings
sudo wget -q https://cyberbotics.com/Cyberbotics.asc

```

2. **Add the Repository:**

```
echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/Cyberbotics.asc] \
https://cyberbotics.com/debian binary-amd64/" | sudo tee \
/etc/apt/sources.list.d/Cyberbotics.list
sudo apt update

```

3. **Install Webots:**

```
sudo apt install webots

```

A.3.2 Method 2: Installing the Debian Package Directly

1. **Download the Debian Package:**
   Go to the official GitHub repository and download the `.deb` file for Webots 2023b.

2. **Install Using Ubuntu Software App:**

   Double-click on the downloaded `.deb` file. This will open it in the Ubuntu Software App. Click on the `Install` button. If Webots is already installed, this button might say `Upgrade` or `Reinstall`. Alternatively, you can use the terminal:

```
sudo apt install ./webots_2023b_amd64.deb

```

   Or use `gdebi` for a potentially smoother installation (install `gdebi` if it's not installed):

```
sudo apt install gdebi
sudo gdebi webots_2023b_amd64.deb

```

A.4 Post-Installation Steps

- **Install Graphics Drivers (if applicable):** For improved performance, especially in 3D simulations, install proprietary NVIDIA or AMD graphics drivers through Ubuntu's "Additional Drivers" settings panel.

- **Install Required Packages for Video Handling:**

```
sudo apt-get update
sudo apt-get install ffmpeg libavcodec-extra
sudo apt-get install ubuntu-restricted-extras

```

B Installation Guide for Webots 2023b on macOS

B.1 Introduction

This guide will help you install Webots 2023b on macOS, detailing both the direct download method using curl to bypass Gatekeeper, and the Homebrew installation method. The guide also addresses special considerations for Apple Silicon Macs.

B.2 Prerequisites

- macOS running on Intel or Apple Silicon processors.
- Terminal access.
- Administrative privileges might be required depending on the installation method.

B.3 Installation Methods

B.3.1 Method 1: Installing from the Installation File

Step 1: Download the Installation File

1. Open Terminal.

2. Use curl to download the Webots disk image. This method helps avoid macOS Gatekeeper interruptions:

```
curl -L -O https://github.com/cyberbotics/webots/releases/download/R2023b/webots-R2023b.dmg
```

3. Mount the downloaded image:

```
open webots-R2023b.dmg
```

Step 2: Install Webots

- To install Webots just for the current user (without needing administrator privileges):

```
mkdir -p ~/Applications
cp -r /Volumes/Webots/Webots.app ~/Applications
```

- To install Webots for all users (administrator privileges required):

```
sudo cp -r /Volumes/Webots/Webots.app /Applications
```

Step 3: Launching Webots

- You can launch Webots using:

```
open ~/Applications/Webots.app  # Using the open command
```

Or directly:

```
~/Applications/Webots.app/webots  # Direct execution
```

- Alternatively, double-click on the Webots icon in the Applications folder.

B.3.2 Method 2: Installing from Homebrew

Step 1: Install Homebrew (if not installed)

1. Open Terminal.
2. Run the following command to install Homebrew:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/released/install.sh)"
```

Step 2: Install Webots

- After installing Homebrew, you can install Webots with:

```
brew install --cask webots
```

B.4 Troubleshooting

Handling macOS Gatekeeper

If you downloaded Webots via a web browser, macOS Gatekeeper might block the app:

1. **Bypassing the Block:**
   - Control-click (or right-click) the Webots app icon.
   - Choose 'Open' from the context menu.
   - Click 'Open' in the dialog that appears to override the Gatekeeper settings.

B.4.1 Apple Silicon Specific Issue

Architecture Compatibility Error:

- If you encounter an error about incompatible architecture on Apple Silicon, ensure Webots runs natively and not under Rosetta:
  - Right-click on Webots in Finder and select 'Get Info'.
  - Ensure that "Open using Rosetta" is unchecked.

C Installation Guide for Webots 2023b on Windows Systems

C.1 Introduction

This guide provides step-by-step instructions on how to install Webots 2023b on Windows operating systems. Webots is a powerful simulation software used for a wide range of robotics applications. The guide covers both regular and silent installation methods.

C.2 Prerequisites

- A Windows computer with administrator privileges
- An internet connection to download the installation file

C.3 Installation Steps

Step 1: Download the Installation File

1. Visit the official Webots website or their GitHub repository to download the latest version of Webots.
2. Download the "webots-R2023b_setup.exe" file.

Step 2: Start the Installation Process

1. Locate the downloaded file in your Downloads folder or wherever you saved it.
2. Double-click on the "webots-R2023b_setup.exe" file to start the installation.

Step 3: Follow the Installation Instructions

1. Once you double-click the installer, follow the on-screen prompts to proceed with the installation. These will guide you through the setup process, including choosing the installation directory.

Step 4: (Optional) Programming Language Setup

- If you plan to use specific programming languages like Python or Java with Webots, follow the additional setup instructions that appear during the installation process to configure these languages properly.

C.4 Silent Installation

For system administrators or users preferring a non-interactive installation, Webots can be installed silently using command-line options:

- Open a Command Prompt with administrator rights.
- Navigate to the folder containing the downloaded setup file.
- Type the following command for a silent installation:

```
    webots-R2023b_setup.exe /SILENT

```

- For a very silent installation (no progress window), use:

```
    webots-R2023b_setup.exe /VERYSILENT

```

C.5 Post-Installation Tips

C.5.1 Upgrade Graphics Drivers

- After installing Webots, if you experience any 3D rendering anomalies or if Webots crashes, it is highly recommended to upgrade your graphics drivers through your graphic card manufacturer's website.

C.5.2 Handling Windows SmartScreen

- If Windows Defender SmartScreen triggers a warning upon launching the Webots installer:
  - Click on "More info".
  - Then, click on "Run anyway" to proceed with the installation.
- Note: Images demonstrating the process are omitted in this LaTeX representation.

C.6 Troubleshooting

- If you encounter issues during the installation:
  - Ensure that the downloaded file is complete and not corrupted.
  - Check if your user account has administrative privileges.
  - Re-download the installer if necessary to avoid issues with incomplete downloads.

D Guide to Adding a Python Controller to a Webots World

D.1 Introduction

This guide is designed to help students in the Digital Control Lab course integrate a Python controller into a pre-existing Webots world. You will learn how to create a Python script that controls a robot within the Webots environment. In this mini-project, a minimal code serving as the robot controller has been provided to students in the attachment, which includes the Maze environment as well. To access this code, if it doesn't open by default upon opening the provided world file, there are two ways to access it. Firstly, you can click on the robot and then right-click to use the "Edit Controller" option. Secondly, you can access the provided code through the `controllers` directory.

D.2 Prerequisites

- Webots simulation software installed.
- Basic knowledge of Python programming.
- Access to the provided Webots world file for the course.

D.3 Step-by-Step Instructions

Step 1: Open Your Webots World

1. **Launch Webots.**
2. **Open the provided world file:** Go to `File > Open World`, and navigate to the location of your world file (‘.wbt‘). Select it and click ‘Open‘.

Step 2: Create a New Python Controller

1. **Add a new controller:** Right-click on the robot node in the scene tree (left panel) and select `Controllers > <new controller...>`.
2. **Name your controller:** In the dialog that appears, type a name for your new controller (e.g., ‘my_python_controller‘) and select Python as the programming language. Click ‘OK‘.

Step 3: Programming Your Controller

1. **Access your controller's directory:** Webots will automatically create a directory with the name you provided (e.g., ‘my_python_controller‘) in your project's ‘controllers‘ directory. Navigate to this directory in your file explorer.
2. **Write your Python script:**
   - Open the ‘my_python_controller.py‘ file created by Webots in a text editor or IDE of your choice.
   - Write or paste your Python code that defines how the robot should behave. Here's a simple example to get started:

```python
from controller import Robot

# create the Robot instance.
robot = Robot()

# get the time step of the current world.
timestep = int(robot.getBasicTimeStep())
```

```
MAX_SPEED = 6.28

# You should insert a getDevice-like function in order to get the
# instance of a device of the robot. Something like:
#  motor = robot.getDevice('motorname')
#  ds = robot.getDevice('dsname')
#  ds.enable(timestep)
leftMotor = robot.getDevice('left wheel motor')
rightMotor = robot.getDevice('right wheel motor')

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

# # set up the motor speeds at 50% of the MAX_SPEED.
leftMotor.setVelocity(.5 * MAX_SPEED)
rightMotor.setVelocity(.5 * MAX_SPEED)

# Main loop:
# - perform simulation steps until Webots is stopping the controller
while robot.step(timestep) != -1:
    # Read the sensors:
    # Enter here functions to read sensor data, like:
    #  val = ds.getValue()

    # Process sensor data here.

    # Enter here functions to send actuator commands, like:
    #  motor.setPosition(10.0)
    pass

# Enter here exit cleanup code.

```

3. **Save your script.**

Step 4: Attach the Controller to Your Robot

1. **Link the controller to your robot:** Return to Webots, and in the scene tree, select your robot.
2. **Set the controller:** In the robot's properties panel, locate the 'controller' field. Click the drop-down menu and select your 'my_python_controller'.
3. **Save the world file:** Ensure all changes are saved in your world file.

Step 5: Run Your Simulation

1. **Start the simulation:** Click the 'Play' button on the Webots toolbar to start the simulation and observe how your robot behaves under the control of your Python script.
2. **Debug if necessary:** If the robot does not behave as expected, stop the simulation, adjust your code, save, and try again.

E Using BFS and DFS for Maze Traversal in Webots with an e-puck Robot

E.1 Introduction

In the field of robotics, autonomous maze navigation is a critical challenge that tests a robot's ability to explore, map, and navigate environments. In this mini-project, you will program an e-puck robot in Webots to traverse and explore every square of a given maze. Two classical algorithms that are particularly useful for this task are Breadth-First Search (BFS) and Depth-First Search (DFS). This section will explain both algorithms and discuss how they can be applied to maze traversal.

E.2 Overview of the Algorithms

E.2.1 Breadth-First Search (BFS)

BFS is an algorithm that explores the maze level by level, progressing uniformly through all possible paths. In the context of maze navigation, BFS will explore all pathways from the starting point, move to all adjacent accessible cells, and then proceed to cells that are two steps away, and so on.

Characteristics of BFS:

- **Complete and optimal:** BFS will find the shortest path to the goal if one exists, as it explores all possible paths from the shortest to the longest.
- **Memory-intensive:** As it stores information about all branching paths, memory usage can be high.
- **Uniform exploration:** BFS is excellent for scenarios where you need to map every part of the maze or find the shortest path to a particular point.

Application in Maze Navigation: BFS can be used to ensure every accessible area of the maze is explored. By treating each intersection or cell in the maze as a node in a graph, BFS can systematically visit every node.

E.2.2 Depth-First Search (DFS)

DFS takes a different approach by exploring as far as possible along a path before backtracking. This means it will follow one path until it can no longer continue, then backtrack to the nearest previous junction where it can try a different path.

Characteristics of DFS:

- **Not necessarily complete or optimal:** DFS does not guarantee the shortest path and can get trapped in loops if not carefully implemented with mechanisms to mark visited paths.
- **Less memory-intensive than BFS:** It only needs to store a stack of nodes along the current path of exploration.
- **Quick to find a path:** While the path may not be the shortest, DFS can quickly find a path to the goal in less structured environments.

Application in Maze Navigation: DFS is suitable for environments where the goal is to reach an endpoint, and the path efficiency is not a priority. In maze exploration, it can be used to quickly map out a path through the maze, though it may not explore all areas unless implemented with additional checks.

E.3 Applying BFS and DFS to Your Project

Conceptual Implementation

1. **Maze Representation:** Represent the maze as a grid or graph where each cell is a node and edges exist between adjacent cells if there is no wall in between.
2. **Starting Point:** Begin at a designated starting point, typically at the entrance of the maze.
3. **Algorithm Execution:**
   - **For BFS:** Implement a queue to hold nodes to be explored. Begin with the starting node, explore all adjacent nodes, and enqueue any node that has not yet been visited.
   - **For DFS:** Use a stack to hold nodes for exploration. Push the starting node, then continuously move to an adjacent unvisited node, pushing it onto the stack. If no unvisited nodes are available, pop the stack to backtrack.

Strategy and Exploration

- **Mapping:** As the robot moves through the maze, use BFS to systematically check and mark each cell, ensuring no areas are left unexplored. This could help in creating a complete map of the maze.
- **Pathfinding:** Use DFS to quickly propose a path through the maze, but be wary of its tendency to choose longer paths or get trapped.

F Overview of the `time` Library in Python

The `time` library helps you work with times, dates, and intervals. It offers functionality to express time in code, ranging from retrieving the current time to executing a pause in the program's execution.

Key Functions in the `time` Library

- `time.time()`: Returns the current time as a floating point number expressed in seconds since the epoch (January 1, 1970, 00:00:00 (UTC)).
- `time.sleep(secs)`: Suspends (delays) execution of the current thread for a given number of seconds.
- `time.localtime()`: Converts a time expressed in seconds since the epoch to a struct_time in local time.

The `time.sleep()` Function

The `time.sleep(secs)` function is particularly useful in both scripts and interactive applications where you need to pause the execution for a certain period. This can be helpful for rate-limiting, adding delays in loops, or simulating real-time operations.

Syntax:

```
time.sleep(seconds)
```

Parameters:

- **seconds**: The number of seconds for which the execution is to be suspended. The argument can be a floating point number for sub-second precision.

Practical Use Cases of `time.sleep()`

Here are some use cases of the `time.sleep()` function, along with explanations:

1. Adding Delays in Loops

Sometimes in a loop, you might want to slow down each iteration to limit how fast the loop executes. This can be useful in scenarios like polling a sensor at fixed intervals.

```
import time

for i in range(5):
    print("Polling sensor...")
    time.sleep(1)  # Delay for 1 second between each sensor poll
```

2. Simulating Real-Time Operations

In simulations or mock-ups of real-world processes, `time.sleep()` can be used to mimic the timing of real-world operations.

```
import time

print("Starting system check...")
time.sleep(2)  # Wait for 2 seconds to simulate the system check
print("System check completed.")
```

3. Controlling Animation Timing

For simple text-based animations or user notifications, `time.sleep()` can help control the display timing.

```
import time

print("Loading ", end="")
for _ in range(5):
    print(".", end="", flush=True)
    time.sleep(0.5)  # Wait half a second before printing the next dot
print(" Done!")
```

4. Rate Limiting in API Calls

When interacting with web APIs, `time.sleep()` can be used to ensure you do not exceed the rate limits imposed by the API provider.

```
import time

for _ in range(5):
    # Call to some API function here
    print("API called")
    time.sleep(1)  # Ensuring at least 1 second between calls
```

G Important Note

In some cases, we rely on introducing delays in our programs in the real world. For example, suppose we want the robot to move forward for three seconds. In practice, with the module mentioned in this section, we first initialize the robot's motors for forward motion, then apply a three-second delay, and finally issue

the stop command. This is also the case in the Webots environment, with the only difference being that the `time.sleep()` command is ineffective here, and we need to create the delay ourselves in the program. The following function achieves this goal:

```
def delay(ms):
    initTime = robot.getTime()      # Store starting time (in seconds)
    while robot.step(timestep) != -1:
        if (robot.getTime() - initTime) * 1000.0 > ms: # If time elapsed (converted into ms)
     is greater than value passed in
            break
```

The following code demonstrates the application of the above function in rotating the robot. You have access to all these components in the provided minimal code.

```
from controller import Robot

# create the Robot instance.
robot = Robot()

# get the time step of the current world.
timestep = int(robot.getBasicTimeStep())
MAX_SPEED = 6.28

# You should insert a getDevice-like function in order to get the
# instance of a device of the robot. Something like:
#  motor = robot.getDevice('motorname')
leftMotor = robot.getDevice('left wheel motor')
rightMotor = robot.getDevice('right wheel motor')


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

# # set up the motor speeds at 50% of the MAX_SPEED.
leftMotor.setVelocity(.5 * MAX_SPEED)
rightMotor.setVelocity(-.5 * MAX_SPEED)
delay(750)
leftMotor.setVelocity(.5 * MAX_SPEED)
rightMotor.setVelocity(.5 * MAX_SPEED)

# Main loop:
# - perform simulation steps until Webots is stopping the controller
while robot.step(timestep) != -1:
    pass
```

Best regards, Setayeshi, Hamidi.