MTRN4110 21T2 Phase A Task Description (Week 1-3)

Released 1/6/2021

1. Overview of the Course Project:

The main project of MTRN4110 21T2 is a simulation-based project adapted from the Micromouse competition. Webots will be used as the simulation platform throughout the project. You will design a mobile robot and implement a controller and a vision program to negotiate a maze autonomously in Webots. The project will contribute 55% to your final mark of this course.

The project consists of four sequential phases, which are connected but attempting one phase is not dependent on the completion of another:

- Phase A: Driving and Perception (week 1-3, 12%, individual)
- Phase B: Path Planning (week 4-6, 12%, individual)
- Phase C: Computer Vision (week 7-9, 12%, individual)
- Phase D: Integration and Improvement (week 10-11, 19%, group)

This document will describe the tasks of Phase A.

2. Overview of Phase A – Driving and Perception:

Phase A aims to build a mobile robot's driving and perception modules for the final maze-solving demonstration. You are required to complete the tasks of this phase by 17:00 Monday Week 4.

2.1. Expectations:

By the end of Phase A, you are expected to have:

- installed Webots on your computer properly;
- completed Webots tutorials (minimum - 1, 2, 4, recommended - 3, 5, 6);
- understood how to run simulations with robots and sensors in Webots;
- been able to create a controller for a robot that can execute a given motion plan;
- been able to detect surrounding objects using onboard sensors of a robot.

2.2. Learning Outcomes Associated with this Assignment:

- **LO1**: Apply relevant theoretical knowledge pertaining to mobile robots, including locomotion, perception and localisation using onboard sensors, navigation and path planning, for practical problem-solving
- **LO3**: Demonstrate practical skills in mechatronics design, fabrication, and implementation

3. Phase A Task Description:

You will be given a Webots world file containing a five by nine maze and an E-puck robot at the beginning of this phase. An example world file is shown in Fig. 1, where the robot is placed at the centre of the top-left corner of the maze and heading towards the south of the view (throughout the course project, we will refer to the top, bottom, left, and right of the maze as North, South, West, and East, respectively).

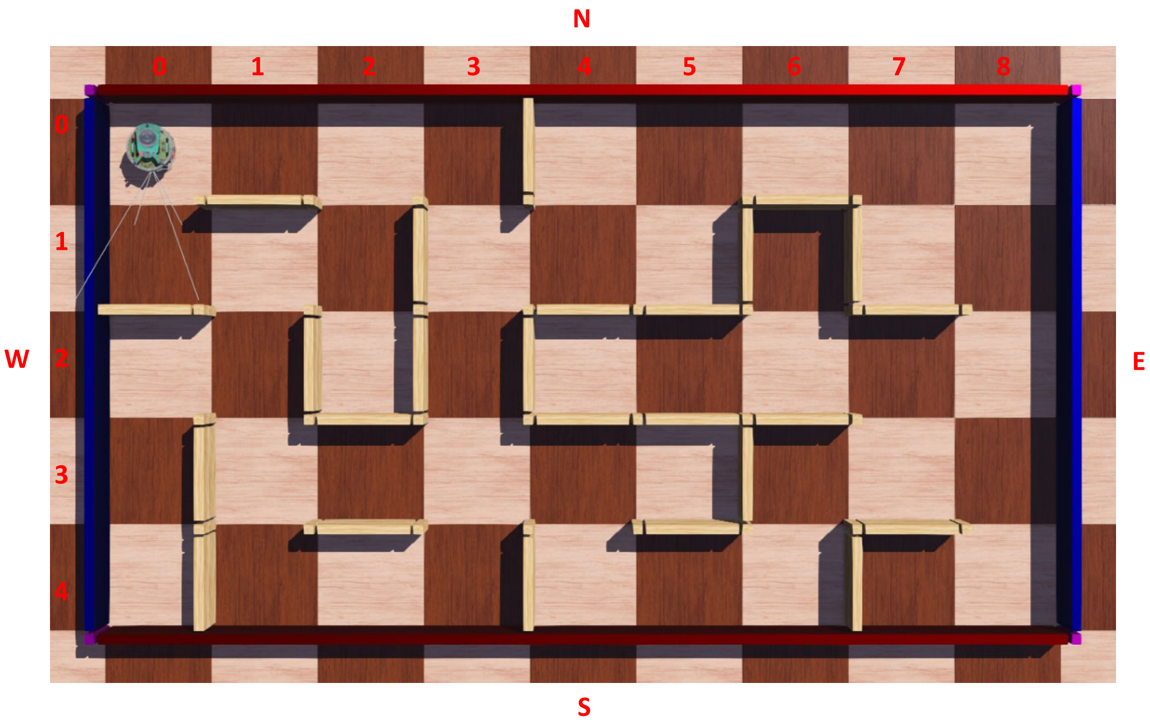


Fig. 1. An example maze layout and an E-puck robot

You must write a controller for the robot, which should complete the following tasks once started.

3.1. Read in a sequence of motion plan from a text file and display it in the console

You will be given a text file named “MotionPlan.txt” containing a sequence of motion commands, e.g.,

Fig. 2. An example motion plan

The motion sequence starts with three characters specifying the initial location and heading of the robot. In the example shown above, the sequence starts with

00S

where the first character (0) stands for the index of the row (0 - 4), the second (0) for the column (0 - 8), and the third (S) for the heading of the robot (N, E, S, W). The top-left corner cell always has an index of (0, 0) for the row and column.

Following the three characters is a sequence of motions represented by a string composed of three characters (F, L, R), where F stands for "Forward for one step", L for "Turn left for 90 deg", and R for "Turn right for 90 deg".

In the example shown above, a sequence that directs the robot from the initial location to the centre of the maze will be (you can validate it by yourself)

FLFFLFRFRFFFLFRFLFFLFRFLFLFFF

In summary, the sequence of motions in the text file will be like the following:

00SFLFFLFRFRFFFLFRFLFFLFRFLFLFFF

The text file given to you will have no spaces or characters other than those mentioned above. Also, note that the initial location and heading will always match the starting status of the robot in the world file given to you, and the motion sequence is always valid.

You should read in the motion plan and display the exact string in the console once the simulation is started.

If you find difficulty implementing reading information from a text file, you can choose to hard-code the motion sequence into your program and forfeit the marks associated with it. In this case, you must define a variable to store the motion sequence at the beginning of your program (so that a tutor can replace it when assessing your work). You should also explicitly indicate you are hard-coding the motion sequence in the Header Comment of your program. Failing to do so would affect the assessment of your submission (incurring a penalty).

In the Header Comment, you should also indicate the platform (Windows/MacOS/Linux) you used to develop your code.

```
/*
 * File:          z1234567_MTRN4110_PhaseA.cpp
 * Date:          XX/XX/XXXX
 * Description:   Controller of E-puck for Phase A - Driving and Perception
 * Author:        XXX
 * Modifications:
 * Platform:      Windows (or MacOS or Linux)
 * Notes:         The motion sequence is hard-coded into the program.
 */
```

Fig. 3. Explicitly indicate hard-coding in the Header Comment of your program

3.2. Display the initial location and heading of the robot, and the existence of the surrounding walls, and write the information into a csv file

You should display the step that the robot is executing. In the initial state, the message should be (using three digits): Step: 000.

After showing the retrieved motion plan, the next step is to parse the information and print the initial location and heading of the robot in the console: Row: 0, Column: 0, Heading: S.

Besides, you need to detect any walls in the robot's front, left, and right. Note that only the walls surrounding the cell that the robot is located in should be considered.

The E-puck robot has some onboard sensors installed by default. You should determine whether these sensors are adequate to detect the walls or add any other sensors provided by Webots as needed (GPS should not be used as it is generally not applicable to indoor scenarios).

Display the existence of the left, front, and right walls. If a wall is detected, print Y; otherwise, print N. In the example of Fig 1, the detected walls should be: Left Wall: N, Front Wall: N, Right Wall: Y

In summary, you should display the following exact message at the initial stage:

Step: 000, Row: 0, Column: 0, Heading: S, Left Wall: N, Front Wall: N, Right Wall: Y

You should also write this information into a csv file named as "MotionExecution.csv" which is stored in the same folder as "MotionPlan.txt". The items should be delimited by commas.

At this step, the csv file should look like this:

	A	B	C	D	E	F	G
1	Step	Row	Column	Heading	Left Wall	Front Wall	Right Wall
2	0	0	0	S	N	N	Y

3.3. Drive the robot following the motion plan

Drive the robot according to the parsed motion plan step by step.

If the motion step to be executed is 'F', move the robot forward for one cell.

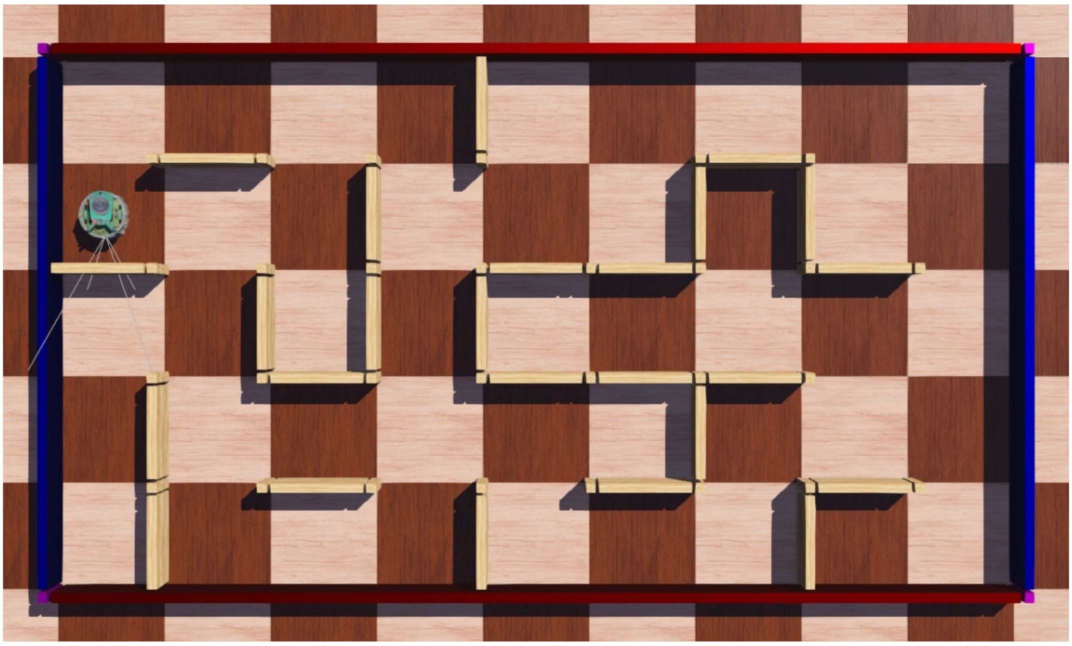


Fig. 4. Move forward for one cell

If the motion step to be executed is 'L', turn the robot 90 deg to its left.

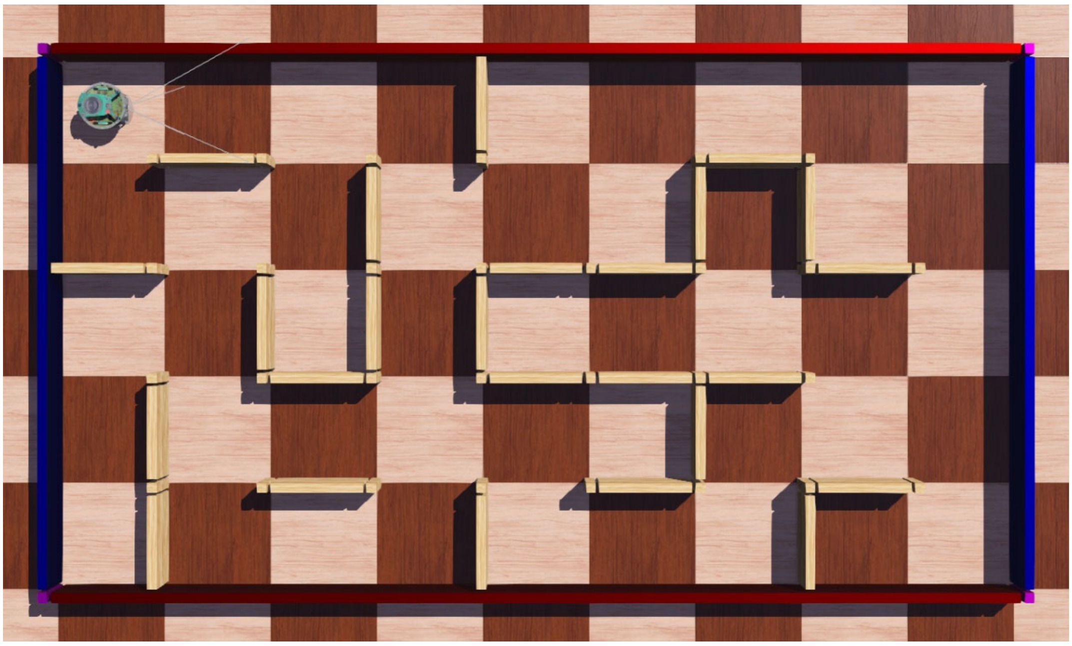


Fig. 5. Turn left for 90 deg

If the motion to be executed step is 'R', turn the robot 90 deg to its right.

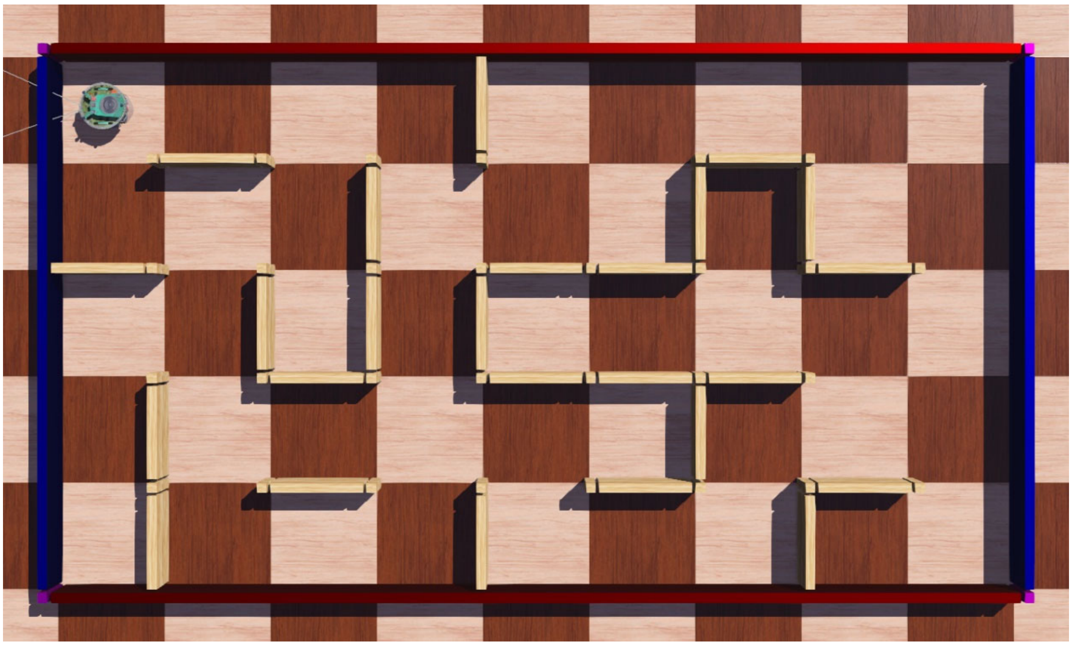


Fig. 6. Turn right for 90 deg

The robot should keep clear of the walls when moving. A penalty will be incurred if the robot hits any walls.

3.4. Display the location and heading of the robot, and the existence of the surrounding walls after each motion step, and write the information into the csv file

Once the robot completes a step, you should display in the console the new location and heading of the robot and the left, front, and right walls of the new cell that the robot stands in.

For example, at the end of the second step, the robot moves to the following location with its heading towards EAST:

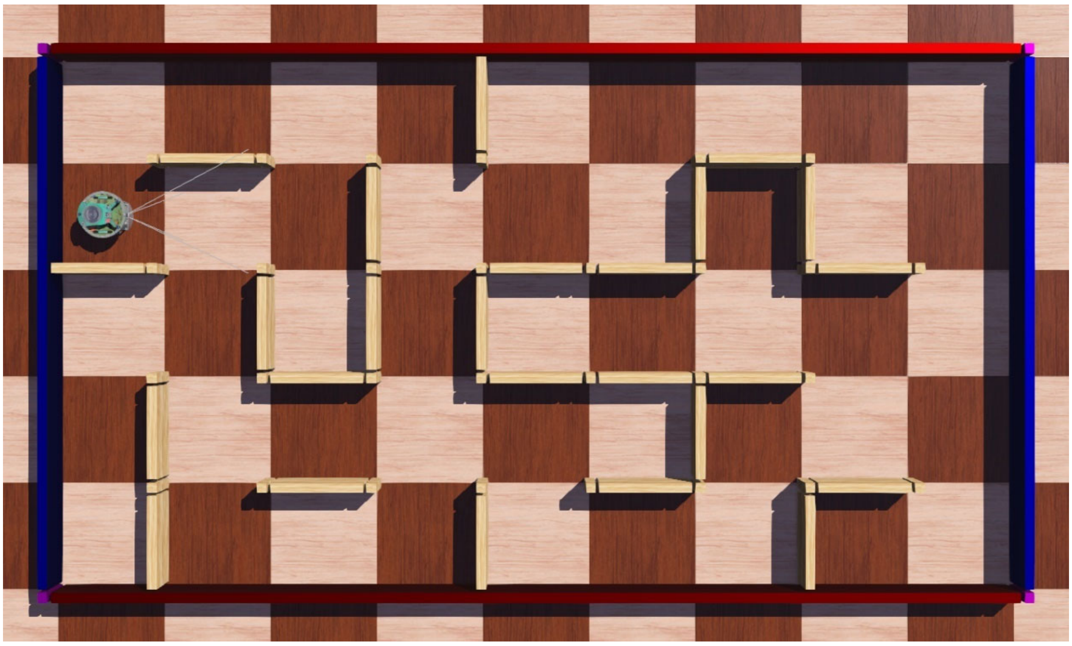


Fig. 7. Example robot location and heading

You should display the following exact message in the console:

Step: 002, Row: 1, Column: 0, Heading: E, Left Wall: N, Front Wall: N, Right Wall: Y

In addition, you should add this information to “MotionExecution.csv”:

	A	B	C	D	E	F	G
1	Step	Row	Column	Heading	Left Wall	Front Wall	Right Wall
2	0	0	0	S	N	N	Y
3	1	1	0	S	N	Y	Y
4	2	1	0	E	N	N	Y

3.5. Repeat tasks 3.3 and 3.4 until all the motion commands are executed

Repeat tasks 3.3 and 3.4 until all the motion commands are executed. Print a message “Motion plan executed!” after the robot completes all the motions.

If the motion plan is:

00SFLFFLFRFRFFFLFRFLFFLFRFLFLFFF

when all the motion commands are executed, the robot should have reached the following state:

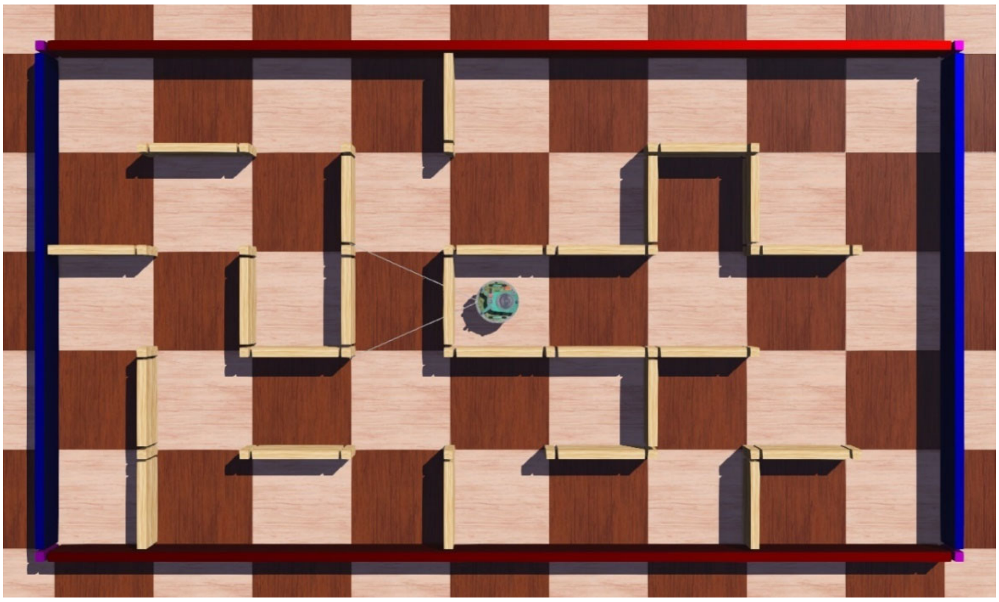


Fig. 8. Robot reaching the centre after execution of all motion commands

And messages printed in the console should be:

```
Console - All
[z1234567_MTRN4110_PhaseA] Reading in motion plan from ../../MotionPlan.txt...
[z1234567_MTRN4110_PhaseA] Motion Plan: 00SFLFFLFRFRFFFLFRFLFFLFRFLFLFFF
[z1234567_MTRN4110_PhaseA] Motion plan read in!
[z1234567_MTRN4110_PhaseA] Executing motion plan...
[z1234567_MTRN4110_PhaseA] Step: 000, Row: 0, Column: 0, Heading: S, Left Wall: N, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 001, Row: 1, Column: 0, Heading: S, Left Wall: N, Front Wall: Y, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 002, Row: 1, Column: 0, Heading: E, Left Wall: N, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 003, Row: 1, Column: 1, Heading: E, Left Wall: Y, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 004, Row: 1, Column: 2, Heading: E, Left Wall: N, Front Wall: Y, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 005, Row: 1, Column: 2, Heading: N, Left Wall: N, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 006, Row: 0, Column: 2, Heading: N, Left Wall: N, Front Wall: Y, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 007, Row: 0, Column: 2, Heading: E, Left Wall: Y, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 008, Row: 0, Column: 3, Heading: E, Left Wall: Y, Front Wall: Y, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 009, Row: 0, Column: 3, Heading: S, Left Wall: Y, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 010, Row: 1, Column: 3, Heading: S, Left Wall: N, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 011, Row: 2, Column: 3, Heading: S, Left Wall: Y, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 012, Row: 3, Column: 3, Heading: S, Left Wall: N, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 013, Row: 3, Column: 3, Heading: E, Left Wall: N, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 014, Row: 3, Column: 4, Heading: E, Left Wall: Y, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 015, Row: 3, Column: 4, Heading: S, Left Wall: N, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 016, Row: 4, Column: 4, Heading: S, Left Wall: N, Front Wall: Y, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 017, Row: 4, Column: 4, Heading: E, Left Wall: N, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 018, Row: 4, Column: 5, Heading: E, Left Wall: Y, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 019, Row: 4, Column: 6, Heading: E, Left Wall: N, Front Wall: Y, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 020, Row: 4, Column: 6, Heading: N, Left Wall: N, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 021, Row: 3, Column: 6, Heading: N, Left Wall: Y, Front Wall: Y, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 022, Row: 3, Column: 6, Heading: E, Left Wall: Y, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 023, Row: 3, Column: 7, Heading: E, Left Wall: N, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 024, Row: 3, Column: 7, Heading: N, Left Wall: N, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 025, Row: 2, Column: 7, Heading: N, Left Wall: N, Front Wall: Y, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 026, Row: 2, Column: 7, Heading: W, Left Wall: N, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 027, Row: 2, Column: 6, Heading: W, Left Wall: Y, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 028, Row: 2, Column: 5, Heading: W, Left Wall: Y, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 029, Row: 2, Column: 4, Heading: W, Left Wall: Y, Front Wall: Y, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Motion plan executed!
```

Fig. 9. Messages displayed after execution of all motion commands

Note that each message should have a prefix "[z1234567_MTRN4110_PhaseA]" where z1234567 is replaced with your zID. Your messages should look exactly the same as shown above.

And "MotionExecution.csv" should be exactly like the following:

	A	B	C	D	E	F	G
1	Step	Row	Column	Heading	Left Wall	Front Wall	Right Wall
2	0	0	0	S	N	N	Y
3	1	1	0	S	N	Y	Y
4	2	1	0	E	N	N	Y
5	3	1	1	E	Y	N	N
6	4	1	2	E	N	Y	N
7	5	1	2	N	N	N	Y
8	6	0	2	N	N	Y	N
9	7	0	2	E	Y	N	N
10	8	0	3	E	Y	Y	N
11	9	0	3	S	Y	N	N
12	10	1	3	S	N	N	Y
13	11	2	3	S	Y	N	Y
14	12	3	3	S	N	N	N
15	13	3	3	E	N	N	N
16	14	3	4	E	Y	N	N
17	15	3	4	S	N	N	N
18	16	4	4	S	N	Y	Y
19	17	4	4	E	N	N	Y
20	18	4	5	E	Y	N	Y
21	19	4	6	E	N	Y	Y
22	20	4	6	N	N	N	Y
23	21	3	6	N	Y	Y	N
24	22	3	6	E	Y	N	N
25	23	3	7	E	N	N	Y
26	24	3	7	N	N	N	N
27	25	2	7	N	N	Y	N
28	26	2	7	W	N	N	Y
29	27	2	6	W	Y	N	N
30	28	2	5	W	Y	N	Y
31	29	2	4	W	Y	Y	Y

3.6. Task summary:

Task	Description
1	Read in a sequence of motion plan from a text file and display it in the console
2	Display the initial location and heading of the robot, and the existence of the surrounding walls, and write the information into a csv file
3	Drive the robot following the motion plan
4	Display the location and heading of the robot, and the existence of the surrounding walls after each motion step, and write the information into the csv file
5	Repeat tasks 3.3 and 3.4 until all the motion commands are executed

4. Specifications and Hints:

4.1. Specifications:

Maze:

1. At the beginning of Phase A, you will be given the same maze layout as shown in the example.
2. The initial location and heading of the robot will also be the same as illustrated.
3. This setup is for your practice. For assessment, you may be tested with a different (but similar) maze layout.
4. The initial location and heading of the robot may also be different and could be at any cell of the maze.
5. The initial location of the robot will always be at the centre of a cell.
6. The initial heading of the robot will always be towards one of the four directions (North, East, South, West).
7. The first three characters in the motion plan given to you will always match the world file.
8. The motion sequence may also be different from the example, but should always be valid (no collision with walls if executed correctly).
9. The maze will always be five by nine and the four borders are always closed.
10. The distance between neighbouring cells is always 0.165 m.
11. The thickness of the walls is always 0.015m.

Robot:

12. For Phase A, you must use E-puck robot for the tasks.
13. You can modify the robot by adding sensors to the <turretSlot> node or the <groundSensorsSlot> node. Note, however, that only the sensors that are provided by Webots are allowed.
14. GPS is not allowed as it is generally not applicable to indoor scenarios.
15. You can modify the maze layout for your practice. But in your submission project, you should not change anything except the sensors of the robot and the controller.
16. The characteristics of E-puck robot can be found here. However, if you are using dead reckoning, the wheel radius and axle length should be corrected as in the following table for accuracy.

Characteristics	Values
Diameter	71 mm
Height	50 mm
Wheel radius	20 mm
Axle Length	56.6 mm
Weight	0.16 kg
Max. forward/backward speed	0.25 m/s
Max. rotation speed	6.28 rad/s

Implementation:

17. You must use C++ to implement the controller if you have taken MTRN2500 Computing for Mechatronic Engineers before.
18. You can choose to use C if you have not taken MTRN2500, but you need to get the lecturer's approval before using it.
19. For portability, you must use the built-in compilers for Windows or macOS.
20. The text file storing the motion sequence should be named "MotionPlan.txt". You should define a path variable at the beginning of your controller program indicating the path of this text file, e.g.,

    ```
    const std::string  MOTION_PLAN_FILE_NAME = "../../MotionPlan.txt";
    ```

    where ../../MotionPlan.txt will allow you to access the file if the folder structure specified in Section 5.1 is followed.
21. The csv file storing the execution information should be named "MotionExecution.csv" and stored in the same folder as "MotionPlan.txt". You should define a path variable at the beginning of your controller program indicating the path of this csv file, e.g.,

    ```
    const std::string MOTION_EXECUTION_FILE_NAME = "../../MotionExecution.csv";
    ```

4.2. Hints:

1. Consult the lecturer/demonstrators if you are unclear about anything.
2. You can use standard printing functions for debugging in Webots. If you want to check the value of a variable during the simulation, you can print it to the console. If you want to see until which line the controller runs successfully, you can also add printing breakpoints at certain steps.
3. Try decreasing the speed of the robot if you use position control mode and the robot does not move to a position as specified.
4. Make the robot stop for a while before reading the sensors to get robust wall detection.
5. You should use forward slash / or double backward slash \\ instead of single backward slash \ to define the path of the file.
6. Implementation of a close-loop controller (such as PID) is recommended.

5. Assessment:

5.1. Submission of your work (tentative as we are exploring options for plagiarism check)

You should zip your project folder and rename it as "z*******_MTRN4110_PhaseA.zip" where z******* is your zID. Submit this zip file to Moodle.

In the folder, you should include both the <worlds> folder and the <controllers> folder.

In the <worlds> folder, you should include your world file, named as "z*******_MTRN4110_PhaseA.wbt" where z******* is your zID.

In the <controllers> folder, you should include your controller folder, named as "z*******_MTRN4110_PhaseA" where z******* is your zID.

You should have the following folder structure in your submission:

```
Z*******_MTRN4110_PhaseA
    |--controllers
    |    |--z*******_MTRN4110_PhaseA
    |        |--build
    |        |--Makefile
    |        |-- z*******_MTRN4110_PhaseA.cpp
    |        |-- z*******_MTRN4110_PhaseA.exe
    |--worlds
    |    |--z*******_MTRN4110_PhaseA.wbt
    |--MotionExecution.csv
    |--MotionPlan.txt
```

These are the essential folders and files you should include. You can also include other files if needed but you should not change the name/location of these folders and files. It is your responsibility to make sure your submission is self-contained.

In addition to the zip file, you need to submit a separate text file which contains all of your code. If you have only one cpp file in your controller, convert it into a text file (.txt) and submit; if you use multiple cpp files, concatenate them into one text file (.txt) and submit. The content of this text file should be the same as the cpp file(s) in your zip file.

5.2. Marking criteria:

This assignment will contribute 12% to your final mark.

You will be assessed five times with different setups. Among them, one test will be on the example given to you for practice. Your final mark will be calculated as the following:

$$mark_{final} = (mark_{example} + mark_{newtest1} + mark_{newtest2} + mark_{newtest3} + mark_{newtest4}) / 5$$

Each attempt will be assessed by using the following criteria.

Task	Description	Marking (out of 100%)

1	Read in a sequence of motion plan from a text file and display it in the console	+10% if all correct, otherwise
		• (8% maximum)
		    ○ +1% each for every 3 consecutive characters correctly displayed. A character is considered correctly displayed if and only if this character and all the preceding characters are correctly displayed.
		• (zero marks)
		    ○ if hard-coding the motion plan into program

2	Display the initial location and heading of the robot, and the existence of the surrounding walls, and write the information into a csv file	+10% if all correct, otherwise • +1% for correctly displaying and writing the initial location and heading • +3% for correctly displaying and writing the left wall • +3% for correctly displaying and writingthe front wall • +3% for correctly displaying and writingthe right wall

3	Drive the robot following the motion plan	+40% if all correct, otherwise • (36% maximum) ○ +2% for each successful move[1]

4	Display the location and heading of the robot, and the existence of the surrounding walls after each motion step, and write the information into the csv file	+40% if all correct, otherwise
		• (18% maximum)
		    ○ +1.5% for correctly displaying and writing robot location and heading after each successful move[1]
		• (18% maximum)
		    ○ +1.5% for correctly displaying and writing detected walls after each successful move[1]

5	Repeat tasks 3.3 and 3.4 until all the motion commands are executed	If any of the above required messages are essentially correct but in a different format from specified, you will only get half of the full marks associated with them.

[1]A move is successful if the robot is correctly and fully moved to a new cell or rotated for 90 deg, without colliding with any walls.

You should make sure your submitted project is self-contained. Demonstrators will only replace the world file and the motion plan file for different tests; no debugging/changes to your code (except in the case of hard-coding the motion plan into the program) should be expected from the assessor. If your code did not run correctly (e.g., crashing after started), you could get zero marks for that test.

5.3. Deadline

The submission will be open from 17:00 AEST 14 June 2021 (Monday Week 3) and the deadline is 17:00 AEST 21 June 2021 (Monday Week 4).

If your assignment is submitted after this date, each 1 hour it is late reduces the maximum mark it can achieve by 2%. For example, if an assignment worth 74% were submitted 10 hours late, the late submission would have no effect. If the same assignment were submitted 30 hours late, it would be awarded 40%, the maximum mark it can achieve at that time.

5.4. Progress Check

You will have your progress checked with your demonstrator in a 5 min meeting between 12:00 and 14:00 on Friday Week 2 (or another time on weekdays Week 2 agreed by you and your demonstrator).

During the session, please show your progress by sharing your screen to your demonstrator. If you don't know how to do this, please watch this short video: https://www.youtube.com/watch?v=ISSDppQnyiE

To pass the progress check, you must demonstrate that you are able to create a controller for the E-puck robot, move the robot forward for one step, then turn the robot left for 90 deg, and display the motion execution messages for steps 000, 001, and 002.

5.5. Plagiarism

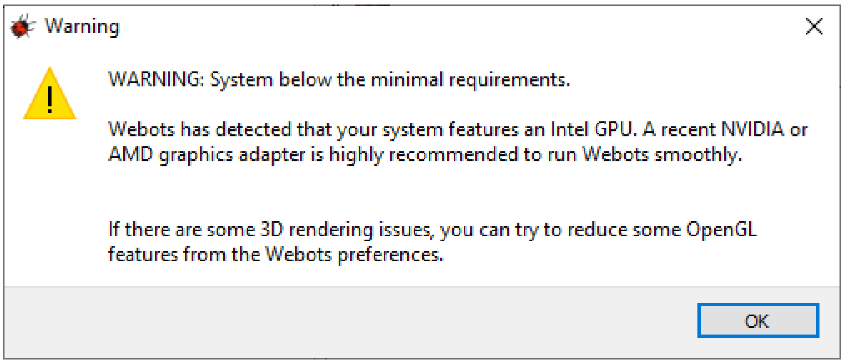
You would get zero marks for the assignment if you were found:

- Knowingly providing your work to anyone and it was subsequently submitted (by anyone), or
- Copying or submitting any other persons' work (except general public open-source libraries/code).

You will be notified and allowed to justify your case before such a penalty is applied.

6. Additional Resources:

- Webots download: https://cyberbotics.com/
  Note that if you encounter this warning when installing Webots, you can just ignore it and the software usually works properly. If you do have issues installing Webots on your computer, you can use any computer in the MECH Computers Lab (Room 204, J17) which have the latest version of Webots installed already.



- Webots user guide: https://cyberbotics.com/doc/guide/index
- Webots reference manual: https://cyberbotics.com/doc/reference/index
- Webots tutorials: https://cyberbotics.com/doc/guide/tutorials
- E-puck: https://cyberbotics.com/doc/guide/epuck
- Webots sensors: https://cyberbotics.com/doc/guide/sensors