MTRN4110 22T2 Phase A Task Description (Week 1-3)

Written By Leo Wu
Updated 3/6/2022: Asking macOS users to configure Makefile to use C++14 (Page 12, Section 4.1.20)
Released 30/5/2022

1. Overview of the Course Project:

The main project of MTRN4110 22T2 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 60% to your final mark in 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, 14%, individual)
- Phase B: Path Planning (Week 4-6, 14%, individual)
- Phase C: Computer Vision (Week 7-9, 14%, individual)
- Phase D: Integration and Improvement (Week 10-12, 18%, 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 must complete the tasks of this phase by 13:00 Monday, Week 4 (20 June).

2.1. Expectations:

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

- installed Webots R2021b 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; and
- been able to output results in the specified format.

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:

At the beginning of this phase, you will be given a Webots world file containing a five by nine maze and an E-puck robot. 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).

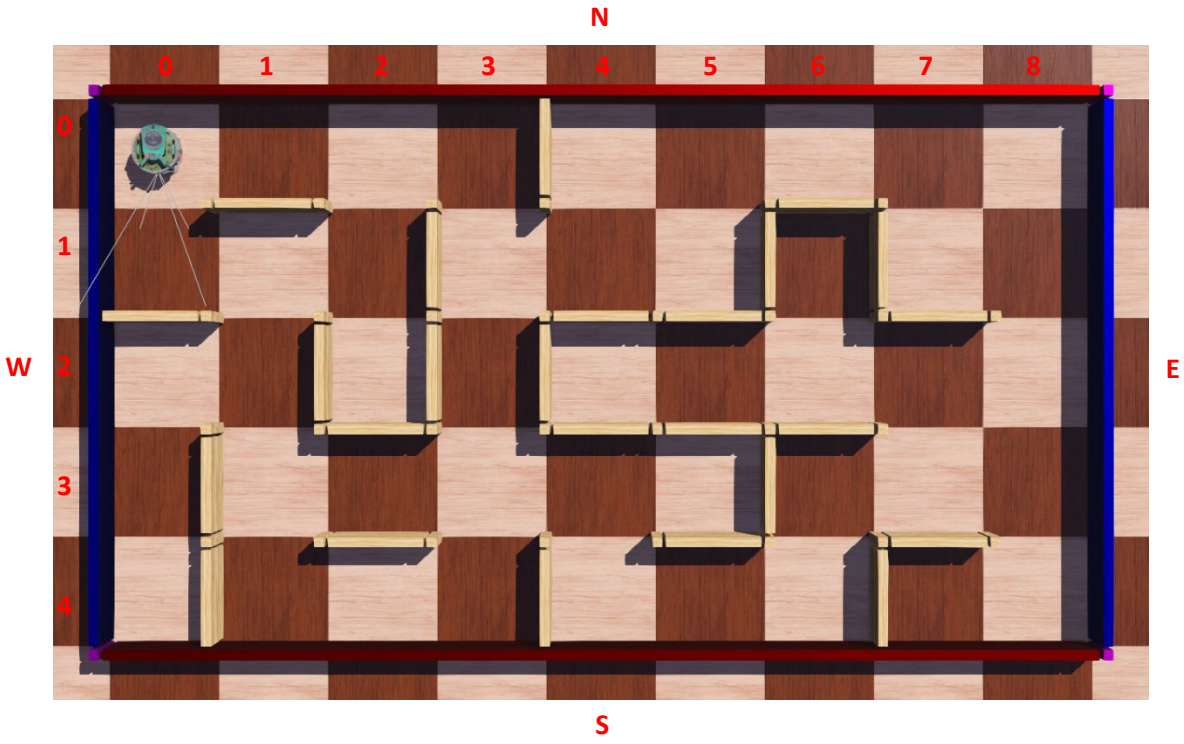


Fig. 1. An example maze layout and an E-puck robot

You must create and implement 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, newlines, or characters other than those mentioned above. Also, note that the initial location and heading will always match the starting state of the robot in the world file given to you, and the motion sequence is always valid (you don't need to check the correctness of the motion sequence or change the robot's initial state).

Your program 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 5% penalty, as if one-day late).

In the Header Comment, you should also indicate the platform (Windows/MacOS/Linux) you used to develop your code so that a tutor can test your submission on your original platform.

```
/*
 * 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

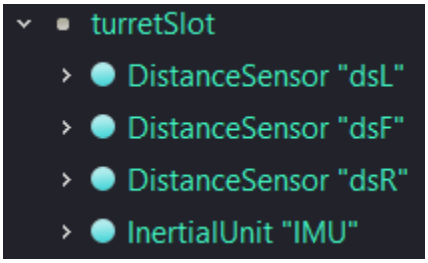
3.2. Display the initial location and heading of the robot and the existence of the surrounding walls, and write the information into a CSV file

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 should detect any walls in the robot's front, left, and right. Only the walls surrounding the cell where the robot is located should be considered.

The E-puck robot has some onboard sensors installed by default. In addition, the robot is added with three Distance Sensors and one Inertial Unit Sensor under the <turretSlot> node. You can use any or all of these four sensors together with E-puck's onboard sensors for the tasks. No other sensors should be added to the robot.



Show 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:

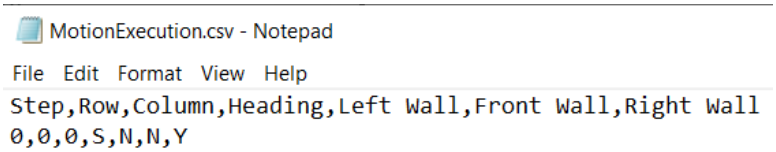
Step: 000, Row: 0, Column: 0, Heading: S, Left Wall: N, Front Wall: N, Right Wall: Y

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

or this if you open it with Notepad:



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.

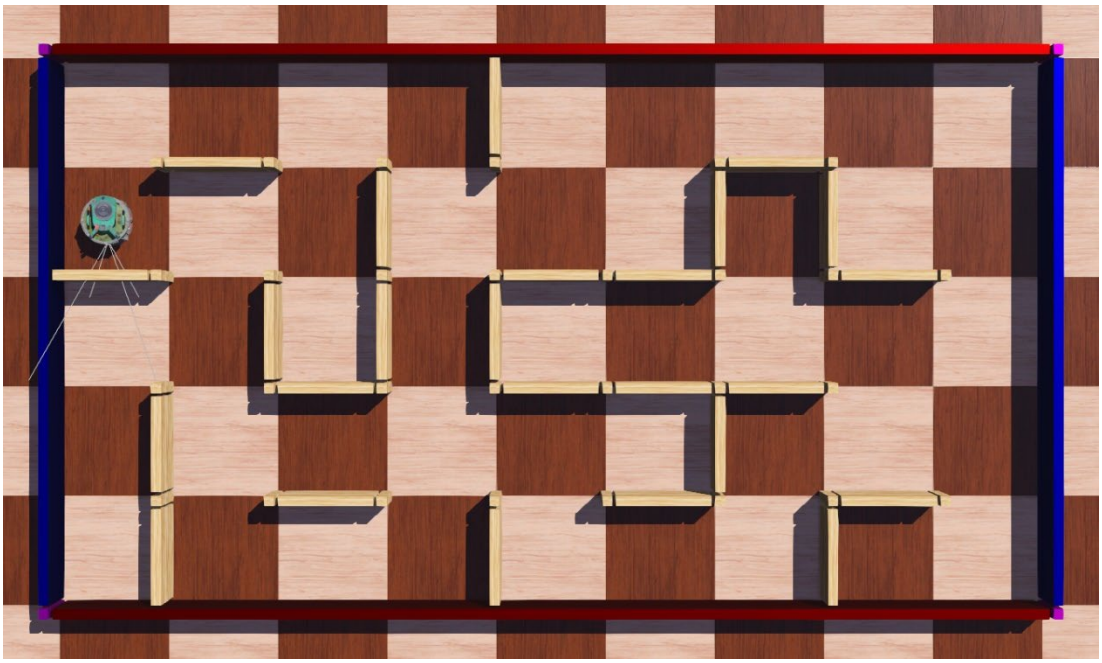


Fig. 4. Move forward for one cell

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

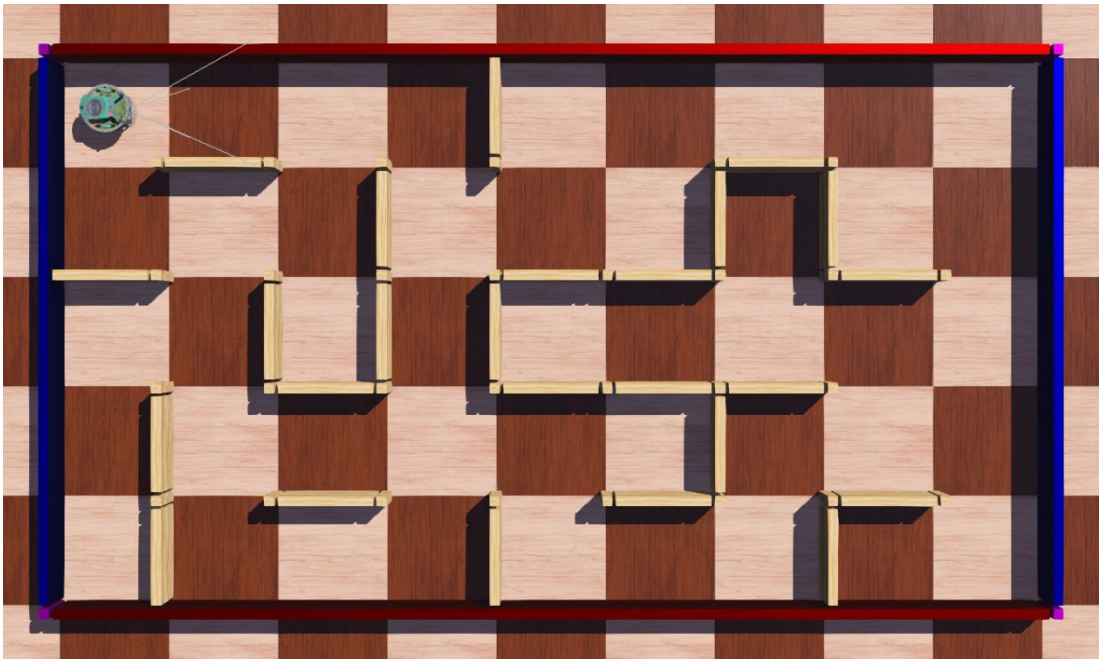


Fig. 5. Turn left for 90 deg

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

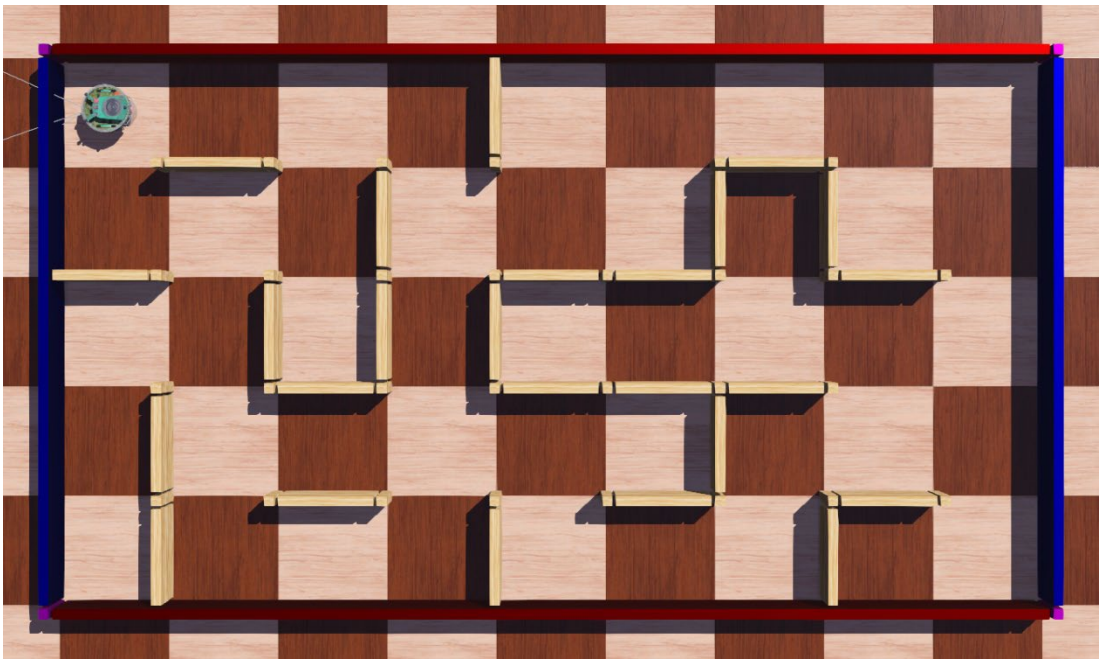


Fig. 6. Turn right for 90 deg

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 show 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:

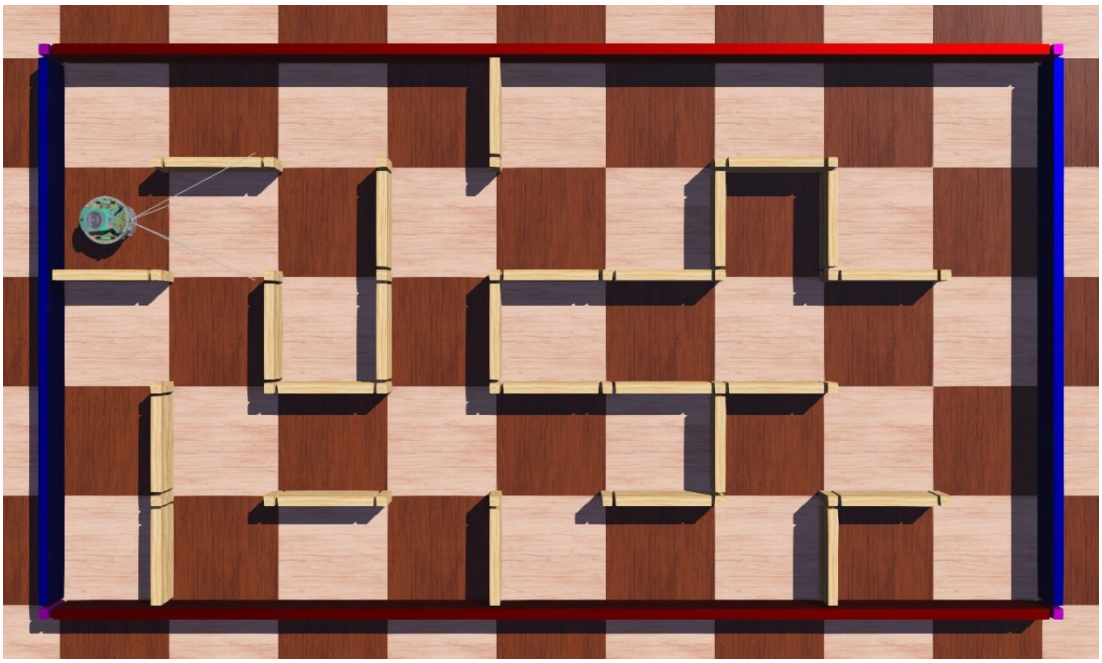


Fig. 7. Example robot location and heading

You should show 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:

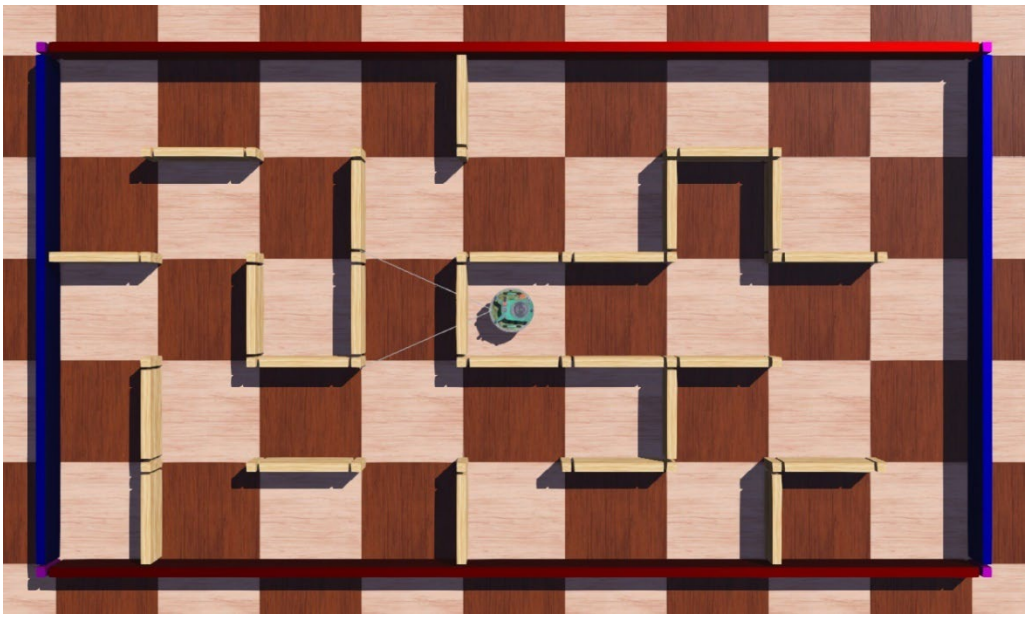


Fig. 8. Robot reaching the centre after execution of all motion commands

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

Fig 10. Content of CSV after execution of all motion commands

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. You should install Webots R2021b; otherwise, there will be some incompatibility issues.
2. At the beginning of Phase A, you will be given the same maze layout as shown in the example.
3. The initial location and heading of the robot will also be the same as illustrated.
4. This setup is for your practice. For assessment, you may be tested with a different (but similar) maze layout.
5. You are expected to test your program with various mazes and motion sequences, including edge cases. **Failing to do so may cause poor performance in some test cases.**
6. The initial location and heading of the robot may also be different and could be at any cell of the maze.
7. The initial location of the robot will always be at the centre of a cell.
8. The initial heading of the robot will always be towards one of the four directions (North, East, South, West).
9. The first three characters in the motion plan given to you will always match the world file.
10. The motion sequence may also be different from the example, but should always be valid (no collision with walls if executed correctly).
11. The maze will always be five by nine, and the four borders are always closed.
12. The distance between neighbouring cells is always 0.165 m.
13. The thickness of the walls is always 0.015m.

Robot:

14. For Phase A, you must use the provided E-puck robot for the tasks and not modify the robot. You will be asked to submit your controller only. **Working on a modified robot during development could lead to incompatibility with the assessor's world file and give you advantages over other students who comply with the constraints, and thus would incur a severe penalty (even getting zero marks).**
15. Three Distance Sensors and one Inertial Unit sensor have been added to the <turretSlot> node. You can use any or all of these sensors together with E-puck's onboard sensors for the tasks. Note all four sensors have noise (expand the feature tree of the world file to see more details); you should carefully handle the noise when using the sensors.
16. You can modify the maze layout for your practice. But you will only submit your controller, which will be tested with the provided world file.
17. The characteristics of the E-puck robot can be found here. However, if you are using dead reckoning, the wheel radius and axle length shown in the table may be inaccurate. You are recommended to tune these parameters using the calibration method introduced during the lecture.

Implementation:

18. You must use C++ to implement the controller if you have taken MTRN2500 Computing for Mechatronic Engineers before.

19. You can choose to use C if you have not taken MTRN2500, but you need to get the lecturer's approval before using it.
20. For portability, you must use the built-in compilers for Windows or macOS.
    - For macOS users, please open the Makefile of your controller and add the following line under the CFLAGS section (CFLAGS = -std=c++14).
    -
```
### ---- C++ Sources ----
### if your program uses several C++ source files:
### CXX_SOURCES = my_plugin.cc my_clever_algo.cpp my_graphics.c++
###
### ---- Compilation options ----
### if special compilation flags are necessary:
CFLAGS = -std=c++14
###
### ---- Linked libraries ----
### if your program needs additional libraries:
### INCLUDE = -I"/my_library_path/include"
### LIBRARIES = -L"/path/to/my/library" -lmy_library -lmy_other_library
```
21. 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.
22. 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";
23. You should use 64 for the TIME_STEP, as used in the Webots Tutorials. No other values should be used.
24. You should make your code modular with good interfaces for the sake of integration in Phase D (refer to the draft specs released).

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 robot's speed 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. Try filtering the sensor readings to get more robust measurements.
6. You should use forward slash / or double backward slash \\ instead of single backward slash \ to define the path of the file.

5. Assessment:

5.1. Submission of your work

During your development, your project folder should look like this:

```
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
```

You should zip (and only zip) your "**controllers**" folder and name it as "z*******_PhaseA_controllers.zip" where z******* is your zID. Submit this zip file to Moodle.

Your submission should only contain one controller, which is the finalised version. It is your responsibility to make sure your submission is self-contained and up-to-date.

As we have developed an automatic tool to do the plagiarism check on your code, it is CRUCIAL that you strictly follow the file structure specified above. Any violation in the submission format would incur a **5% penalty (as if one-day late)**, as that adds much more work to the assessors (we will still do the plagiarism check!).

5.2. Marking criteria:

This assignment will contribute 14% 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[1], 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[1], otherwise • (8% maximum): ○ +1% for correctly displaying and writing the initial location and heading to CSV ○ +3% for correctly displaying and writing the left wall to CSV ○ +3% for correctly displaying and writing the front wall to CSV ○ +3% for correctly displaying and writing the right wall to CSV
3	Drive the robot following the motion plan	+40% if all correct, otherwise • (36% maximum): ○ X / N * 40%, where X is the number of cells traversed with successful move[2] and N is the total number of cells traversed by the given 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	+40% if all correct[1], otherwise • (36% maximum): ○ Y / M * 40%, where Y is the number of correct values[3] for location/heading/walls and M is the total number of values for location/heading/walls expected from the given motion plan
5	Repeat tasks 3.3 and 3.4 until all the motion commands are executed	

[1]correct means the messages (both value and format) in console and CSV are exactly the same as required (see Fig. 9 and 10). A message that is essentially correct but in the wrong format will be deemed incorrect. Note in particular: step, row, and column should start from 0; the order of the items should not be changed; the letters used should be the same as specified.

[2]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.

[3]A value is correct if the robot's current and past moves are successful as defined above and the reporting of the location/heading/walls at the current cell is correct.

You should make sure your submitted project is self-contained and up-to-date. 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 starting), you could get **zero** marks for that test.

5.3. Deadline

The submission will be open from 13:00 AEST 14 June 2022 (Monday Week 3) and the deadline is 13:00 AEST 20 June 2022 (Monday Week 4).

We will apply a university-wide late policy that has been specified by UNSW and MME (refer to the course outline):

Late policy

Work submitted late without an approved extension by the course coordinator or delegated authority is subject to a late penalty of five percent (5%) of the maximum mark possible for that assessment item, per calendar day.

The late penalty is applied per calendar day (including weekends and public holidays) that the assessment is overdue. There is no pro-rata of the late penalty for submissions made part way through a day. This is for all assessments where a penalty applies.

Work submitted after five days (120 hours) will not be accepted and a mark of zero will be awarded for that assessment item.

For example:

- Your course has an assessment task worth a total of 100 marks.
- You submit the assessment 2 days (or part thereof) late (i.e. from 24-48 hours after the deadline).
- The submission is graded and awarded a mark of 65/100.
- A late penalty of 10 marks is deducted from your awarded mark (2 days @ 5% of 100 marks).
- Your adjusted final score is 55/100.

Note the 5% penalty will be multiplied by the maximum possible mark and applied directly to the **awarded mark** (refer to the example).

Students are expected to manage their time to meet deadlines or request extensions (apply for special consideration) as early as possible before the deadline.

5.4. Progress Check

You will have your progress checked with your demonstrator in a 5 min meeting in your Week 2 workshop session.

During the session, please show your progress to your demonstrator in person. For the online workshop, please share your screen with your demonstrator. If you don't know how to do this, please watch this short video: How to share your screen in a Microsoft Teams meeting

To pass the progress check, you must demonstrate you can create a controller for the robot, move the robot forward for one step, turn the robot left for 90 deg, and print the locations and headings of the robot (no sensing is required). These should be completed in one program run.

The progress check makes up 1% of the overall course mark (included in the 14%).

All progress checks should be completed before the end of the scheduled workshop session. The late submission policy does not apply to progress checks.

5.5. Plagiarism

If you are unclear about the definition of plagiarism, please refer to What is Plagiarism? | UNSW Current Students.

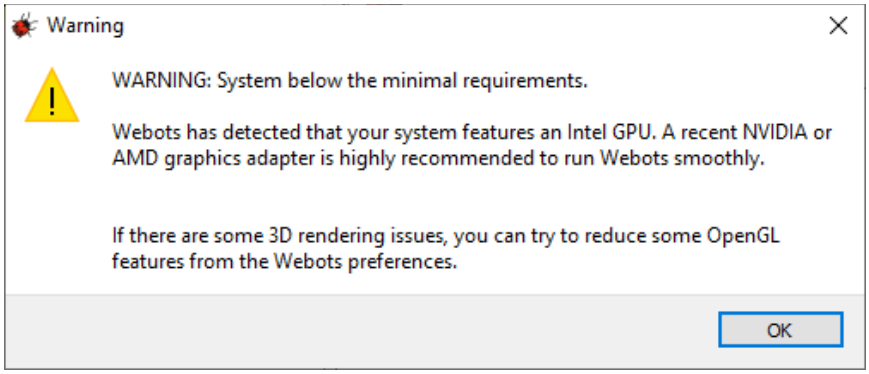
You would get zero marks for the assignment if you were found:

- Knowingly providing your work to anyone and it was subsequently submitted (by anyone), or
- Copying or submitting any other persons' work, **including code from previous students of this course** (except general public open-source libraries/code, for which you should explicitly cite the source wherever applicable).

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 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 203, J17) which has 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