# MTRN2500 20T3 C++ Assignment Task Descriptions

# (Updated 17/10/2020)

## Changelog

- 17/10: 1) Deleted "[To Check: what is the extension on MacOS?]." in Section 2.2.1 on Page 4 and Section 2.3.1 on Page 6; 2) Added Point 8 to Section 3.1 on Page 13; 3) Added "NOT" to Item "Good documentation and comments" in Table 4 on Page 16
- 08/10: 1) Corrected cut-off images of Section 2.1 on Page 3; 2) Changed Supervisor node to Field node of Section 2.5 on Page 9; 3) Added Point 7 to Section 3.1 on Page 13
- 05/10: 1) First release

## 1. Overview of the C++ Assignment:

The main assignment of the C++ part of MTRN2500 20T3 is a robotics-simulation project based on the Webots simulator. You will be given a Webots world file containing two robots and a racing setup. You are required to develop ONE controller using C++ that enables BOTH robots to complete a series of tasks in the given setup.

The assignment is worth 35% of your total marks in this course. The marks will include two components: 25 marks for the functionality of your program and 10 marks for the quality. The due date of this assignment is 12:00 29th Oct 2020 (Midday Thursday Week 7). You should complete this assignment individually.

### 1.1. Expectations:

By the end of the assignment, you are expected to have been able to:

- understand the basic features of C++ and use them appropriately in developing a robot controller,
- use Object-Oriented programming techniques to design a C++ program,
- make appropriate use of common standard libraries (including STL),
- be familiar with file I/O, and
- follow good programming styles and practices.

### 1.2. Learning Outcomes Associated with this Assignment:

- **LO1:** Be well versed with structured and modular programming using C/CPP and to have appreciated the use of software to communicate with external devices.
- **LO2**: Be able to understand how to interface to an external device through a computer program to effect control action.
- **LO3**: Be able to develop prototype user interfaces to assist in the development of controlled Mechatronic systems.

## 2. Task Descriptions:

In this assignment you will be given a Webots world file as shown in Fig. 1. The setup contains a set of walls and two robots. The robot on the left has the name "ROBOT_RED" and the one on the right is named "ROBOT_BLUE". Both robots have the same characteristics except for the name and colour (see section 3.1).

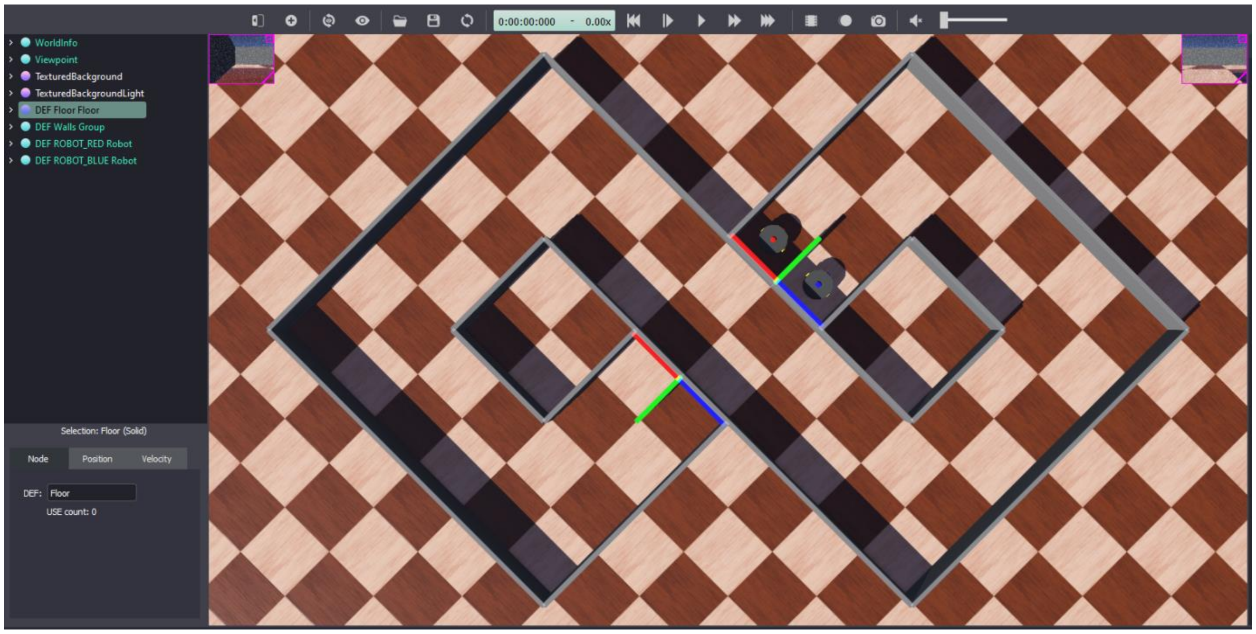


Fig. 1. The setup of the assignment (MTRN2500.wbt)

You are required to develop ONE controller, named "Robot_Controller_z1234567", that will be added to BOTH robots. Here z1234567 should be replaced by your zID. The controller should complete the following tasks:

### 2.1. Display a list of commands at the start of the simulation (1 mark):

Once the simulation is started, the controller should display a list of commands in the console (you do not need to follow the indentation shown in the example):

```
TASK_MANAGER: Please select the command:
TASK_MANAGER: [1] run ROBOT_RED in manual-control mode and write data to
                  ROBOT_RED.csv
TASK_MANAGER: [2] run ROBOT_BLUE in manual-control mode and write data to
                  ROBOT_BLUE.csv
TASK_MANAGER: [3] run ROBOT_RED and ROBOT_BLUE in wall-following mode
TASK_MANAGER: [4] run ROBOT_RED or ROBOT_BLUE in shortest-time mode
TASK_MANAGER: [5] read data from ROBOT_RED.csv and process
TASK_MANAGER: [6] read data from ROBOT_BLUE.csv and process
```

The controller should then wait for the user's input. The user will need to click on the simulation window (so that the window can capture keyboard messages) and press a key. The user may input any key at this stage:

- If the user input was '1', the controller should display the following message and execute the tasks defined in section 2.2

```
TASK_MANAGER: Your input was 1 - now run ROBOT_RED in manual-control mode and
write data to ROBOT_RED.csv
```

- If the user input was '2', the controller should display the following message and execute the tasks defined in section 2.3

```
TASK_MANAGER: Your input was 2 - now run ROBOT_BLUE in manual-control mode and
write data to ROBOT_BLUE.csv
```

- If the user input was '3', the controller should display the following message and execute the tasks defined in section 2.4

```
TASK_MANAGER: Your input was 3 - now run ROBOT_RED and ROBOT_BLUE in wall-
following mode
```

- If the user input was '4', the controller should display the following message and execute the tasks defined in section 2.5

```
TASK_MANAGER: Your input was 4 - now run ROBOT_RED or ROBOT_BLUE in shortest-time
mode
```

- If the user input was '5', the controller should display the following message and execute the tasks defined in section 2.6

```
TASK_MANAGER: Your input was 5 - now read data from ROBOT_RED.csv and process
```

- If the user input was '6', the controller should display the following message and execute the tasks defined in section 2.7

```
TASK_MANAGER: Your input was 6 - now read data from ROBOT_BLUE.csv and process
```

- If the user input was any key else, the controller should display nothing

Note 1, the above applies to the controllers of both robots (both robots will be added with the same controller). Therefore, there could be duplicate messages displayed by the TASK_MANAGER of the two controllers as both controllers can capture the key input from the user.

Note 2, throughout the program, it is fine if there are duplicate messages output by TASK_MAGAGER because two identical controllers are running simultaneously.

### 2.2. Run ROBOT_RED in manual-control mode and write data to ROBOT_RED.csv (3 marks):

In this subtask, the controller should allow the user to control ROBOT_RED manually using the keyboard.

Once entering this mode, if the robot is ROBOT_RED, the controller should display the following messages in the console (if the robot is ROBOT_BLUE, the controller should do nothing until the end of the program):

```
TASK_MANAGER: Your input was 1 - now run ROBOT_RED in manual-control mode and
              write data to ROBOT_RED.csv
ROBOT_RED: Starting writing data to ROBOT_RED.csv
ROBOT_RED: Please use the following commands to control the motion:
ROBOT_RED: [W]     Move forward
ROBOT_RED: [S]     Move backward
ROBOT_RED: [A]     Turn left
ROBOT_RED: [D]     Turn right
ROBOT_RED: [SPACE] Stop
```

The controller should now start writing data to "ROBOT_RED.csv" and wait for the user's input.

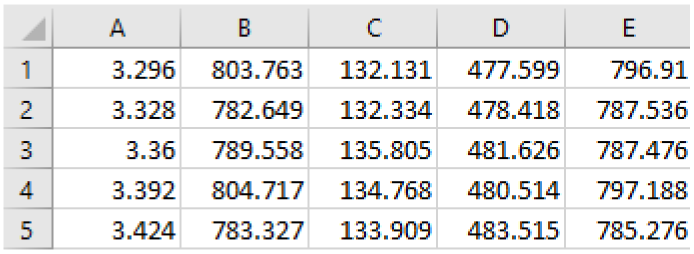
### 2.2.1. Write data to ROBOT_RED.csv

The controller should write the following data to the file:

Virtual_Time,Raw_Reading_ds0,Raw_Reading_ds1,Raw_Reading_ds2,Raw_Reading_ds3

Here Virtual_Time is the simulated time of a running simulation (link), and the raw readings are the raw measurement values generated by the four distance sensors of the robot.

Example data stored in ROBOT_RED.csv are shown in Fig. 2. The data are separated by commas. There is no need to add a header to the first line.



| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | 3.296 | 803.763 | 132.131 | 477.599 | 796.91 |
| 2 | 3.328 | 782.649 | 132.334 | 478.418 | 787.536 |
| 3 | 3.36 | 789.558 | 135.805 | 481.626 | 787.476 |
| 4 | 3.392 | 804.717 | 134.768 | 480.514 | 797.188 |
| 5 | 3.424 | 783.327 | 133.909 | 483.515 | 785.276 |

Fig. 2. Example data stored in ROBOT_RED.csv

The file should be stored in the same folder that contains "Robot_Controller_z1234567.exe"/"Robot_Controller_z1234567". The file should be opened in "append" mode, i.e., adding new data to the end if the file exists or create a new file if it does not exist.

### 2.2.2. Run ROBOT_RED in manual-control mode

While the controller writes data to ROBOT_RED.csv, the controller should also wait for user's input to manually control the robot:

- When the user clicks on the simulation window and presses the 'W/w' key, the robot should move forward until a further and different command is received.
- When the user clicks on the simulation window and presses the 'S/s' key, the robot should move backward until a further and different command is received.

- When the user clicks on the simulation window and presses the 'A/a' key, the robot should turn left until a further and different command is received.
- When the user clicks on the simulation window and presses the 'D/d' key, the robot should turn right until a further and different command is received.
- When the user clicks on the simulation window and presses the 'SPACE' key, the robot should stop until a further and different command is received.

In all the cases above, the speed of the motors can be decided by you. However, the speed of the motors should never exceed the maximum speed (in both directions) as specified in section 3.1 at any time.

An example scenario when ROBOT_RED is run in manual-control mode is shown in Fig. 3.

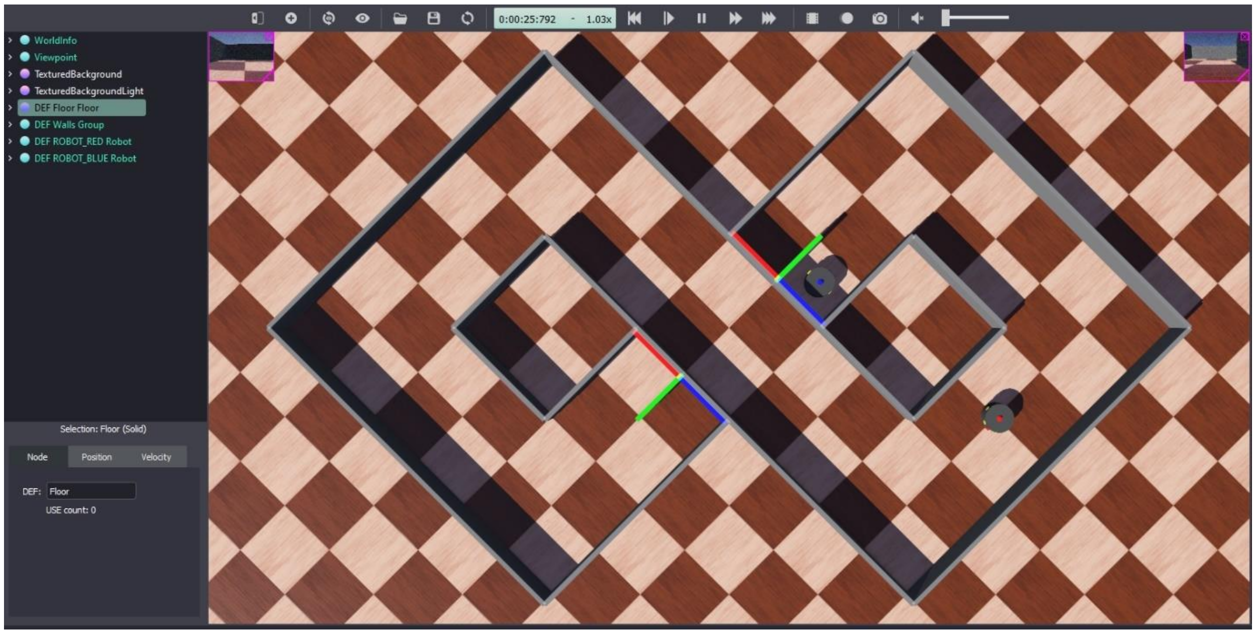


Fig. 3. An example scenario when ROBOT_RED is run in manual-control mode

### 2.2.3. Termination of the mode

The user will terminate the mode by ending the whole simulation (e.g., clicking the pause button and resetting the simulation). You do not need to detect whether the robot has reached the target or not in this mode. You do not need to return to the initial state after the mode is terminated by the user.

## 2.3. Run ROBOT_BLUE in manual-control mode and write data to ROBOT_BLUE.csv (3 marks):

In this subtask, the controller should allow the user to control ROBOT_BLUE manually using the keyboard.

Once entering this mode, if the robot is ROBOT_ BLUE, the controller should display the following messages in the console (if the robot is ROBOT_ RED, the controller should do nothing until the end of the program):

```
TASK_MANAGER: Your input was 2 - now run ROBOT_BLUE in manual-control mode and
              write data to ROBOT_BLUE.csv
ROBOT_BLUE: Starting writing data to ROBOT_BLUE.csv
ROBOT_BLUE: Please use the following commands to control the motion:
ROBOT_BLUE: [UP]    Move forward
ROBOT_BLUE: [DOWN]  Move backward
ROBOT_BLUE: [LEFT]  Turn left
ROBOT_BLUE: [RIGHT] Turn right
ROBOT_BLUE: [SPACE] Stop
```

The controller should now start writing data to "ROBOT_BLUE.csv" and wait for the user's input.

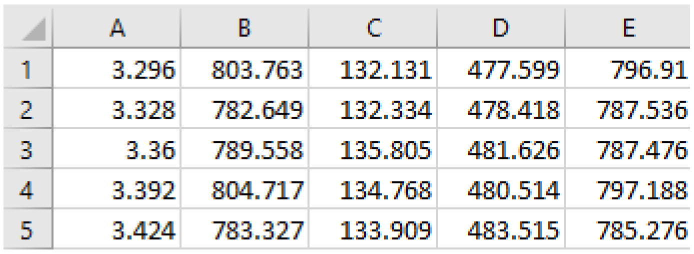
### 2.3.1. Write data to ROBOT_BLUE.csv

The controller should write the following data to the file:

Virtual_Time,Raw_Reading_ds0,Raw_Reading_ds1,Raw_Reading_ds2,Raw_Reading_ds3

Here Virtual_Time is the simulated time of a running simulation (link), and the raw readings are the raw measurement values generated by the four distance sensors of the robot.

Example data stored in ROBOT_BLUE.csv are shown in Fig. 4. The data are separated by commas. There is no need to add a header to the first line.



|   | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | 3.296 | 803.763 | 132.131 | 477.599 | 796.91 |
| 2 | 3.328 | 782.649 | 132.334 | 478.418 | 787.536 |
| 3 | 3.36 | 789.558 | 135.805 | 481.626 | 787.476 |
| 4 | 3.392 | 804.717 | 134.768 | 480.514 | 797.188 |
| 5 | 3.424 | 783.327 | 133.909 | 483.515 | 785.276 |

Fig. 4. Example data stored in ROBOT_BLUE.csv

The file should be stored in the same folder that contains "Robot_Controller_z1234567.exe"/"Robot_Controller_z1234567". The file should be opened in "append" mode, i.e., adding new data to the end if the file exists or create a new file if it does not exist.

### 2.3.2. Run ROBOT_BLUE in manual-control mode

While the controller writes data to ROBOT_BLUE.csv, the controller should also wait for user's input to manually control the robot:

- When the user clicks on the simulation window and presses the 'UP' key, the robot should move forward until a further and different command is received.
- When the user clicks on the simulation window and presses the 'DOWN' key, the robot should move backward until a further and different command is received.
- When the user clicks on the simulation window and presses the 'LEFT' key, the robot should turn left until a further and different command is received.
- When the user clicks on the simulation window and presses the 'RIGHT' key, the robot should turn right until a further and different command is received.

- When the user clicks on the simulation window and presses the 'SPACE' key, the robot should stop until a further and different command is received.

Note, Webots has defined the above four control keys as webots::Keyboard::UP, webots::Keyboard::DOWN, webots::Keyboard::LEFT, webots::Keyboard::RIGHT, respectively.

In all the cases above, the speed of the motors can be decided by you. However, the speed of the motors should never exceed the maximum speed (in both directions) as specified in section 3.1 at any time.

An example scenario when ROBOT_BLUE is run in manual-control mode is shown in Fig. 5.

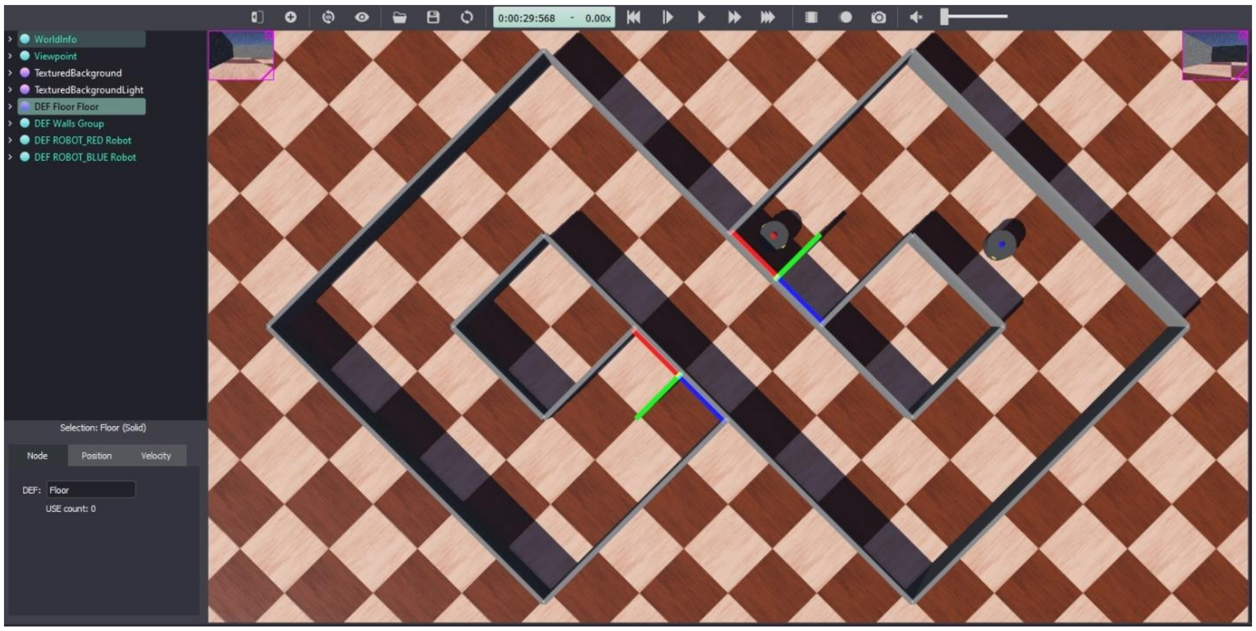


Fig. 5. An example scenario when ROBOT_BLUE is run in manual-control mode

### 2.3.3. Termination of the mode

The user will terminate the mode by ending the whole simulation (e.g., clicking the pause button and resetting the simulation). You do not need to detect whether the robot has reached the target or not in this mode. You do not need to return to the initial state after the mode is terminated by the user.

## 2.4. Run ROBOT_RED and ROBOT_BLUE in wall-following mode (8 marks):

In this subtask, the controller should control BOTH robots to run automatically from the start position to the respective target position using a wall-following strategy. The controller should show the following messages at the start of this mode (the starting time is the Virtual Time when the robot starts moving; it is fine if there are duplicate messages from TASK_MANAGER)

```
TASK_MANAGER: Your input was 3 - now run ROBOT_RED and ROBOT_BLUE in wall-
              following mode
ROBOT_RED: Starting wall-following mode at 3.296 s
ROBOT_BLUE: Starting wall-following mode at 3.296 s
```

The controller should adopt a wall-following strategy, i.e., ROBOT_RED should always follow the walls on its left, and ROBOT_BLUE should always follow the walls on its right. The two robots should automatically stop at the target positions illustrated in Fig. 6, respectively. Along the way, the two robots should always remain in their respective lane and do not cross the middle line.

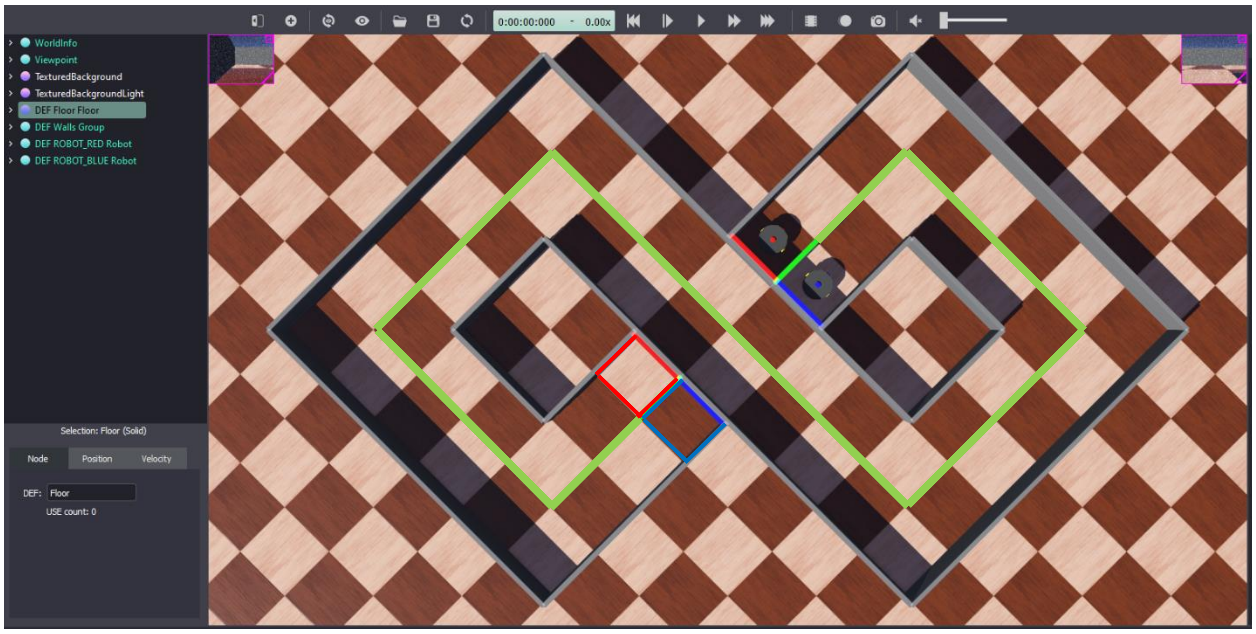


Fig. 6. Lanes and target positions for the two robots (illustrative, not marked in the world file)

An example scenario when both robots are run in wall-following mode is shown in Fig. 7.

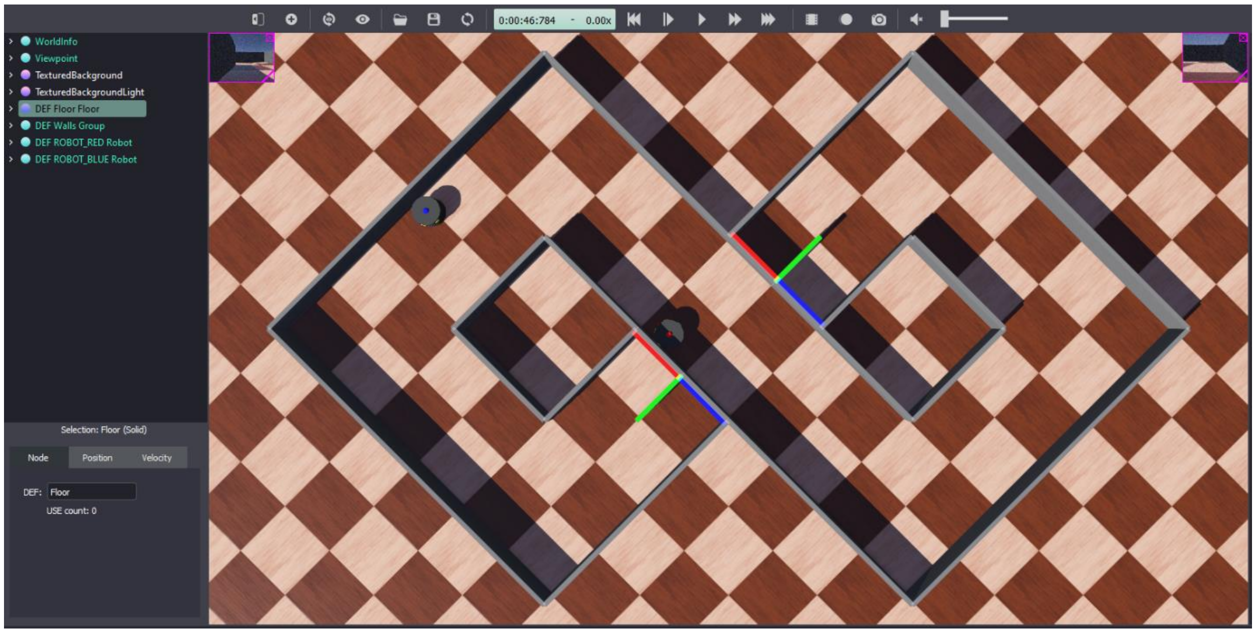


Fig. 7. An example scenario when both robots are run in wall-following mode

An example scenario when both robots reach the target position is shown in Fig. 8.

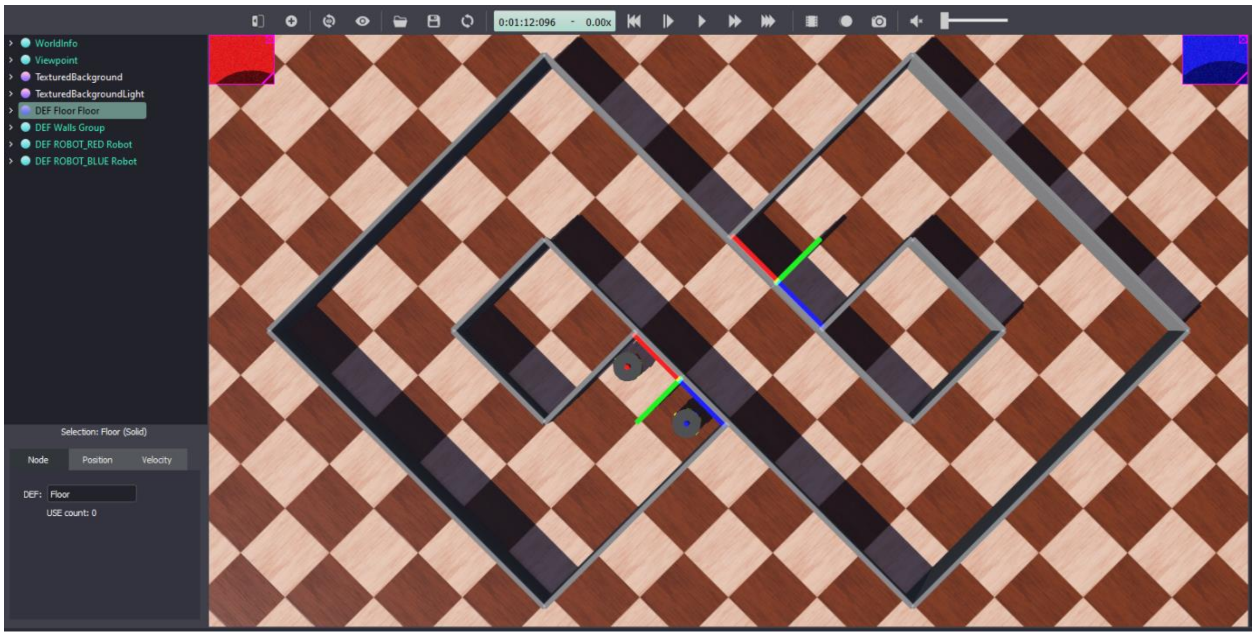


Fig. 8. An example scenario when both robots reach the target position

Once the robot reaches the target position (fully within the target square), the robot should stop its motion and immediately report the virtual time:

```
ROBOT_RED: Reaching target position in wall-following mode at 51.328 s
ROBOT_BLUE: Reaching target position in wall-following mode at 51.328 s
```

The controller should not display any other message during the subtask. The user will terminate the mode by ending the whole simulation (e.g., clicking the pause button and resetting the simulation). You do not need to return to the initial state after the mode is terminated by the user.

### 2.5. Run ROBOT_RED or ROBOT_BLUE in shortest-time mode (4 marks):

In this subtask, the controller should control EITHER robot to run automatically from the start position to the respective target position using the shortest-time strategy. You can choose which robot to run in this mode and set it in the code. The other robot should do nothing until the end of the program. The controller should show the following messages at the start of this mode (the starting time is the Virtual Time when the robot starts moving; here suppose ROBOT_RED is set to run in this mode)

```
TASK_MANAGER: Your input was 4 - now run ROBOT_RED or ROBOT_BLUE in shortest-time
              mode
ROBOT_RED: Starting shortest-time mode at 3.296 s
```

The controller can adopt ANY strategy, including wall-following, to achieve the shortest-time. However, the controller should NOT use any Webots node that directly enables the acquisition/manipulation of the location of the robot (e.g., the Field node). There is no constraint that the robot must remain in its lane in this mode (i.e., the robot can cross the middle line). The robot should reach the target position automatically and report the virtual time immediately when it arrives at the target:

```
ROBOT_RED: Reaching target position in shortest-time mode at 43.520 s
```

An example scenario when ROBOT_RED reaches the target position in shortest-time mode is shown in Fig. 9.

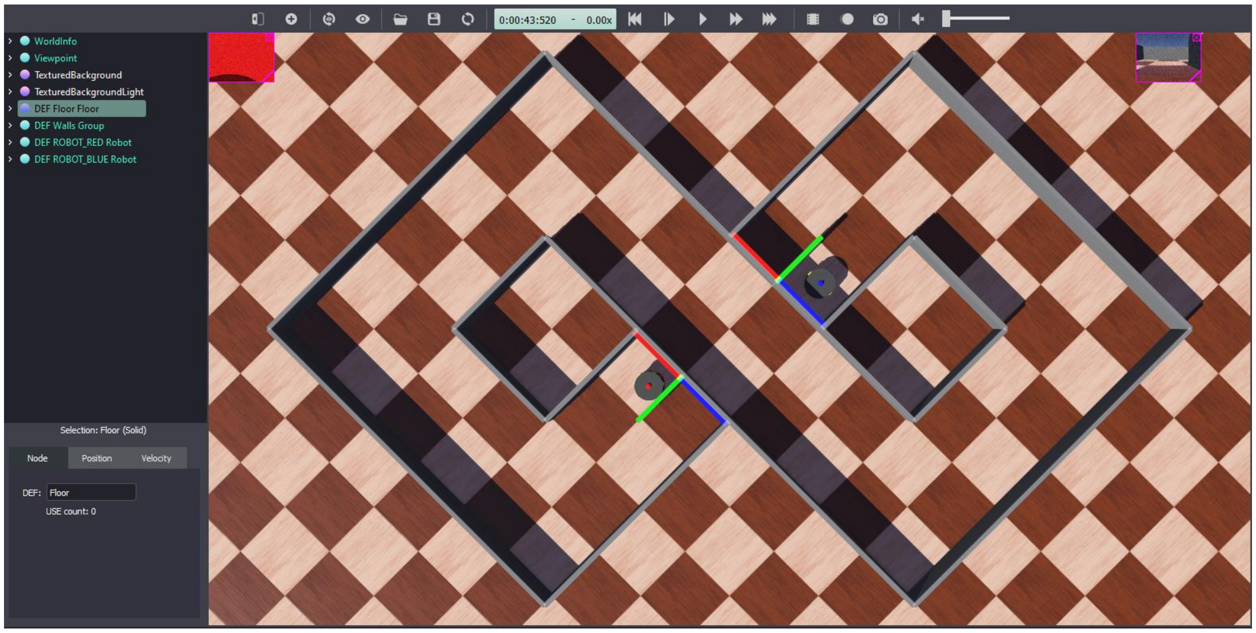


Fig. 9. An example scenario when ROBOT_RED reaches the target position in shortest-time mode

The controller should not display any other message during the subtask. The user will terminate the mode by ending the whole simulation (e.g., clicking the pause button and resetting the simulation). You do not need to return to the initial state after the mode is terminated by the user.

## 2.6. Read data from ROBOT_RED.csv and process (3 marks):

In this subtask, the controller should read all the data from ROBOT_RED.csv and process the data. If the robot is ROBOT_RED, the controller should display a message (if the robot is ROBOT_BLUE, it should do nothing until the end of the program):

```
TASK_MANAGER: Your input was 5 - now read data from ROBOT_RED.csv and process
```

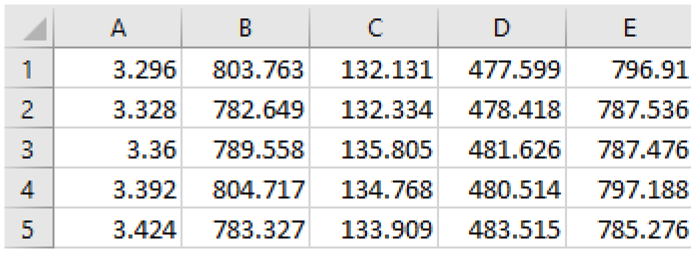
The controller should then read all the data into the program (e.g., a 2D vector of double). If reading was successful, display:

```
ROBOT_RED: Reading data from ROBOT_RED.csv succeeded!
```

Otherwise (e.g., the file did not exist), display:

```
ROBOT_RED: Reading data from ROBOT_RED.csv failed!
```

If reading was successful, calculate the maximum, minimum, and average of each column of the data. For example, if the data stored in ROBOT_RED were:



| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | 3.296 | 803.763 | 132.131 | 477.599 | 796.91 |
| 2 | 3.328 | 782.649 | 132.334 | 478.418 | 787.536 |
| 3 | 3.36 | 789.558 | 135.805 | 481.626 | 787.476 |
| 4 | 3.392 | 804.717 | 134.768 | 480.514 | 797.188 |
| 5 | 3.424 | 783.327 | 133.909 | 483.515 | 785.276 |

Fig. 10. Example data stored in ROBOT_RED.csv

Then display the following messages (all the data displayed should have three digital numbers for the fraction part):

```
ROBOT_RED: The maximum values are – 3.424, 804.717, 135.805, 483.515, 797.188
ROBOT_RED: The minimum values are – 3.296, 782.649, 132.131, 477.599, 785.276
ROBOT_RED: The average values are – 3.360, 792.803, 133.789, 480.334, 790.877
```

The user will terminate the mode by ending the whole simulation (e.g., clicking the pause button and resetting the simulation). You do not need to return to the initial state after the mode is terminated by the user.

## 2.7. Read data from ROBOT_BLUE.csv and process (3 marks):

In this subtask, the controller should read all the data from ROBOT_BLUE.csv and process the data. If the robot is ROBOT_BLUE, the controller should display a message (if the robot is ROBOT_RED, it should do nothing until the end of the program):

```
TASK_MANAGER: Your input was 6 - now read data from ROBOT_BLUE.csv and process
```

The controller should then read all the data into the program (e.g., a 2D vector of double). If reading was successful, display:

```
ROBOT_BLUE: Reading data from ROBOT_BLUE.csv succeeded!
```

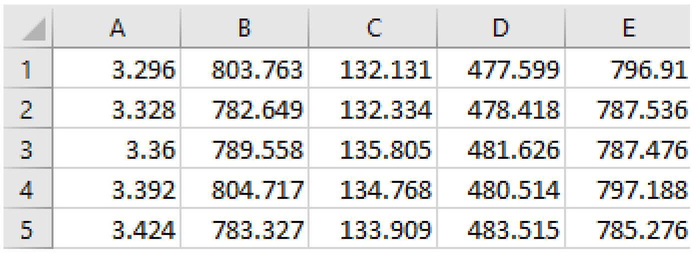
Otherwise (e.g., the file did not exist), display:

```
ROBOT_BLUE: Reading data from ROBOT_BLUE.csv failed!
```

If reading was successful, sort the data based on the last column (from smallest to largest). Store the new data to a new file “ROBOT_BLUE_sorted.csv” and display a message:

```
ROBOT_BLUE: Writing data to ROBOT_BLUE_sorted.csv succeeded!
```

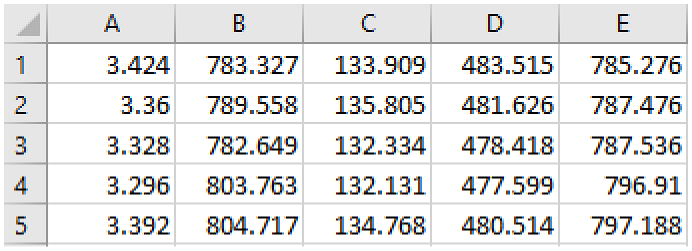
For example, if the data stored in “ROBOT_BLUE.csv” were:



|   | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | 3.296 | 803.763 | 132.131 | 477.599 | 796.91 |
| 2 | 3.328 | 782.649 | 132.334 | 478.418 | 787.536 |
| 3 | 3.36 | 789.558 | 135.805 | 481.626 | 787.476 |
| 4 | 3.392 | 804.717 | 134.768 | 480.514 | 797.188 |
| 5 | 3.424 | 783.327 | 133.909 | 483.515 | 785.276 |

Fig. 10. Example data stored in ROBOT_BLUE.csv

Then the data written to "ROBOT_BLUE_sorted.csv" should be:



|   | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | 3.424 | 783.327 | 133.909 | 483.515 | 785.276 |
| 2 | 3.36 | 789.558 | 135.805 | 481.626 | 787.476 |
| 3 | 3.328 | 782.649 | 132.334 | 478.418 | 787.536 |
| 4 | 3.296 | 803.763 | 132.131 | 477.599 | 796.91 |
| 5 | 3.392 | 804.717 | 134.768 | 480.514 | 797.188 |

Fig. 11. Example data written to ROBOT_BLUE_sorted.csv

The user will terminate the mode by ending the whole simulation (e.g., clicking the pause button and resetting the simulation). You do not need to return to the initial state after the mode is terminated by the user.

## 2.8. Task summary:

Table 1. Task summary

| **Task** | **Description** |
|---|---|
| 1 | Display a list of commands at the start of the simulation (1 mark) |
| 2 | Run ROBOT_RED in manual-control mode and write data to ROBOT_RED.csv (3 marks) |
| 3 | Run ROBOT_BLUE in manual-control mode and write data to ROBOT_BLUE.csv (3 marks) |
| 4 | Run ROBOT_RED and ROBOT_BLUE in wall-following mode (8 marks) |
| 5 | Run ROBOT_RED or ROBOT_BLUE in shortest-time mode (4 marks) |
| 6 | Read data from ROBOT_RED.csv and process (3 marks) |
| 7 | Read data from ROBOT_BLUE.csv and process (3 marks) |

## 3. Specifications and Hints:

### 3.1. Specifications:

1. The Webots world file provided with the description will also be used for assessment. You should NOT change the world file.
2. You should create ONE controller named “Robot_Controller_z1234567” where z1234567 is your zID and add the controller to BOTH robots for test.
3. Both robots have the SAME characteristics as follows:

Table 2. Characteristics of both robots

| Devices | Characteristics | Values |
|---|---|---|
| Motor | Left Motor Name | “left wheel motor” |
| | Right Motor Name | “right wheel motor” |
| | Maximum Speed | 10 rad/s |
| Distance Sensor | Distance Sensor Number | 4 |
| | Distance Sensor Name | “ds0”, “ds1”, “ds2”, “ds3” |
| | Obstacle detection threshold | around 800.0 (tunable) |
| Camera | Camera Name | “camera” |
| | Image width | 80 pixels |
| | Image height | 60 pixels |

4. The distribution of the devices on the robot:

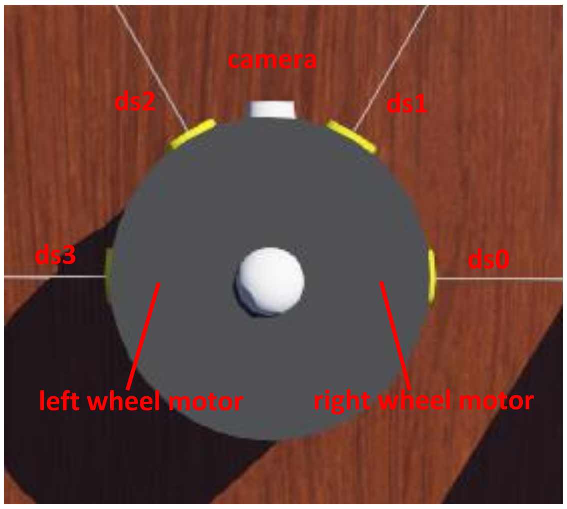


Fig. 12. Device distribution

5. Ideally you should separate implementations and interfaces into multiple cpp and hpp files. However, for the sake of assessment, you are required to implement the program in ONE cpp file.
6. Your controller should use C++14 as the standard of C++ version.
7. The controller should NOT use any Webots node that directly enables the acquisition/manipulation of the location of the robot (e.g., the Field node).
8. Throughout the program, the speed of the motors should NOT exceed the maximum speed (in both directions) as specified in section 3.1 at any time.

## 3.2. Hints:

1. Consult the lecturer/demonstrators if you are unclear about anything.
2. When asking for help, post your error messages so that the problems are better understood
3. Refer to the example projects provided by Webots
4. Refer to the hpp and cpp files provided by Webots
5. Have a careful thought on the whole structure of the program
6. Divide the problem into subproblems and solve them one by one
7. Try using the concepts of object-oriented programming (classes, inheritance, etc.) as much as possible
8. If the structure is not clear at the beginning, try writing your program in a procedural structure first, and then optimise your program by restructuring/refactoring, etc.
9. Follow the style guide as much as possible (style matters!)

## 4. Assessment:

### 4.1. Marking criteria:

This assignment will contribute 35% to your final mark. The marks will include two components: 25 marks for the functionality of your program and 10 marks for the quality.

#### 4.1.1. Functionality (25 marks):

The functionality of your program will be marked by following the criteria below:

Table 3. Marking of the functionality (25 marks)

| Task | Description | Marking |
|---|---|---|
| 1 | Display a list of commands at the start of the simulation (1 mark) | +0.5 if the display is correct<br>+0.5 if the response of key input is correct |
| 2 | Run ROBOT_RED in manual-control mode and write data to ROBOT_RED.csv (3 marks) | +1.5 if remote control is correct<br>+1.5 if writing data is correct |
| 3 | Run ROBOT_BLUE in manual-control mode and write data to ROBOT_BLUE.csv (3 marks) | +1.5 if remote control is correct<br>+1.5 if writing data is correct |
| 4 | Run ROBOT_RED and ROBOT_BLUE in wall-following mode (8 marks) | +2 if ROBOT_RED wall following is correct<br>+2 if ROBOT_RED reaching target is correct<br>+2 if ROBOT_BLUE wall following is correct<br>+2 if ROBOT_BLUE reaching target is correct |
| 5 | Run ROBOT_RED or ROBOT_BLUE in shortest-time mode (4 marks) | The marking of this section is explained in section 4.1.1.1 |
| 6 | Read data from ROBOT_RED.csv and process (3 marks) | +1 if reading and error handling is correct<br>+2 if max, min, and average are correct |
| 7 | Read data from ROBOT_BLUE.csv and process (3 marks) | +1 if reading and error handling is correct<br>+2 if sorting and writing are correct |

##### 4.1.1.1. Marking of task 5

Marking of this component will be based on the performance of each submission against the best of all the submissions.

Suppose $t_{best}$ is the best (shortest) time achieved from all the submissions, $t_{base} = 30\ s$ is the baseline, and $t_{sub}$ is the time achieved by a submission. The mark for the submission $m_{sub}$ will be calculated by:

$$t_{sub} = \min\left(t_{sub}, t_{base}\right)$$

$$m_{sub} = \frac{t_{base} - t_{sub}}{t_{base} - t_{best}} \times 4\ marks$$

As two extreme examples, the best submission ($t_{sub} = t_{best}$) will get full 4 marks, while a submission having a time worse than the baseline ($t_{sub} \geq t_{base}$) in the shortest-time mode gets 0 marks.

#### 4.1.2. Quality (10 marks):

The quality of your code will be marked by following the criteria below. Note that the mark for this part will be capped by a proportional mark of your performance in the functionality. For example, if your mark for functionality is X (out of 25) marks, then the maximum mark you can get for the quality is X/25*10 marks (make sure your program works!).

Table 4. Marking of the quality (10 marks)

| **Items** | **Description** | **Marking** |
|---|---|---|
| Good format | Neat and tidy code. Consistent and appropriate use of indentation, braces, spaces, etc. | 1 |
| Good documentation and comments | Functions should have comments document its interface. Critical or complex section of code should have comment explaining the logic to future maintainers of your code. Comments should explain the thinking behind the code and NOT just restating the code that is obvious. Trivial/obvious items do not need to be commented. | 1 |
| Good choice of names | Good variable names should answer the important questions another developer might have about the variable or function. | 1 |
| Use of functions | Consistent use of functions to break program into small fragments will make the program more manageable, easier to understand, and avoid repeating yourself (DRY principle). | 1 |
| Use of constants | Magic values in the code should be replaced by constants to both give a name to the value and make changing the value in the future easier. Reference parameters and member functions should be made constant wherever possible to avoid unintended modifications. | 1 |
| Use of references and auto | References are used in function parameters where it is appropriate to avoid overhead of cloning. Auto is used for long types to improve readability. | 1 |
| OOP design | Design of classes and hierarchies is rational and appropriate, achieving good modularity, reusability, data hiding etc. | 4 |

### 4.2. Submission of your work

You should save a copy of the cpp file of your controller and rename it as "MTRN2500_Robot_Controller_z1234567.cpp" where z1234567 is your zID.

Submit this cpp file (only this file) via Moodle.

### 4.3. Submission deadline

The submission deadline is 12:00 AEST 29 Oct 2020 (Midday Thursday Week 7).

If your assignment is submitted after this time, each 1 hour it is late reduces the maximum mark it can achieve by 1% (out of 100%). 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 70%, the maximum mark it can achieve at that time.

## 4.4. Progress check

You will have your progress checked with your demonstrator in a 5 min meeting on the afternoon of Friday Week 5 (or another time on weekdays Week 5 by appointment; you should contact your demonstrator to make an appointment in advance).

To pass the progress check, you must demonstrate that you are able to display the messages in task 1 AND manually control the movement of at least one of the robots in task 2/3 (no need to show data writing).

## 5. Additional Resources:

- Webots user guide: https://cyberbotics.com/doc/guide/index
- Webots reference manual: https://cyberbotics.com/doc/reference/index
- Webots official tutorial 1: https://cyberbotics.com/doc/guide/tutorial-1-your-first-simulation-in-webots
- Webots official tutorial 4: https://cyberbotics.com/doc/guide/tutorial-4-more-about-controllers
- Webots robot node: https://www.cyberbotics.com/doc/reference/robot
- Webots motor node: https://cyberbotics.com/doc/reference/motor
- Webots distance sensor node: https://cyberbotics.com/doc/reference/distancesensor
- Webots camera node: https://www.cyberbotics.com/doc/reference/camera
- Webots keyboard node: https://www.cyberbotics.com/doc/reference/keyboard
- Webots virtual time: https://www.cyberbotics.com/doc/reference/glossary
- Example Webots project: Webots->Menu->File->Open Sample World->languages->cpp->example.wbt
- MTRN2500 Style Guide: https://moodle.telt.unsw.edu.au/pluginfile.php/6101784/mod_resource/content/1/StyleGuide.pdf
- Sort 2D vectors on basis of a particular column: https://www.geeksforgeeks.org/sorting-2d-vector-in-c-set-1-by-row-and-column/