Assignment 2 - Webots

MTRN2500 - UNSW School of Mechanical and Manufacturing Engineering

November 4, 2023

Changelog

- 27/10/23: Initial release
- 04/11/23: Update to file layout options for project

1 Introduction

In scenarios such as autonomous robotic swarm search and rescue operations or swarm inspection tasks, a fleet of robots may be deployed to conduct individual inspections of various targets before providing valuable reports on the targets condition. For these applications, smaller, quicker and more efficient robots are used to inspect the environment, where a larger more capable robot can then move optimally to a desired target rather than inspect all targets.

This assignment simulates such a scenario, in which three agile 2-wheeled Scout Robots and a larger, more capable Leader Robot are tasked with identifying, exploring, and reporting information on different Objects of Interest (OOIs) within the Webots simulator.

In this assignment, you will receive a Webots world file that includes four robots and three Objects of Interest (OOIs). Your task is to develop multiple controllers using C++ to enable these robots to autonomously locate and navigate towards the OOIs represent by large coloured cylinders.

2 Learning Outcomes

- LO1: Demonstrate effective use of CPP programming to solve problems in relation to Mechatronic applications.
- LO2: Recognise and follow modern best practices in CPP programming.
- LO3: Explain the concepts within Object-Oriented Programming and apply these concepts to C++ program design.

3 Brief Task Description

3.1 Basic task

The Leader Robot equipped with a long-range 2D LiDar scans the environment for Objects of Interest (OOIs) at the start of a simulation. It does not move at this stage and will be able to see all the OOI from its initial position. For each OOI detected, the Leader sends a wireless command to each of the three Scout Robots containing their target position. Each Scout Robot moves to within 0.5 meter of the OOI and reports back its colour to the Leader. At this point the Scout can stop moving and stay in front of the target. Once the green OOI is found (there is always one green OOI and the rest will be red), the Leader Robot moves to within 1 meter of it. At this point the Leader Robot stops, concluding the run. You do not need to exit out of the simulation at the conclusion of the run. During the simulation, the Leader Robot will log all important communication and events to an output text file to be reviewed later.

The project sample code includes three world files, with a basic one shown below in Figure 1.

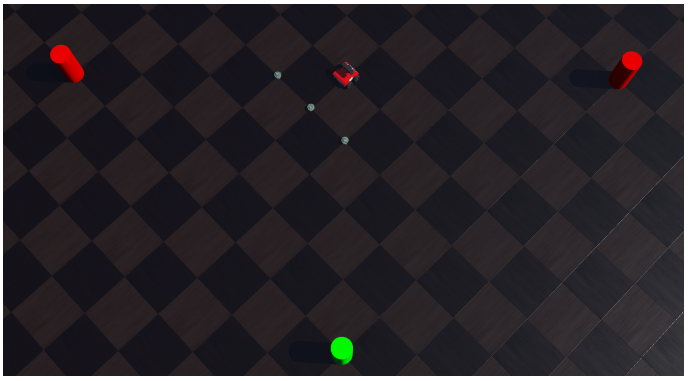

Figure 1: Supplied world environment.

The OOIs are scattered throughout the area. These are tall coloured pillars, one of which will be green OOI and the remainder red. For any initial configuration of a world environment, every OOI will be visible to the LiDar, No OOI will be hidden or obscured behind another OOI. The target position of each OOI extracted from the Leader Robot's LiDar does not have to be in the centre of the cylinder; any point within 0.2m radius of the centre of the OOI will be accepted.

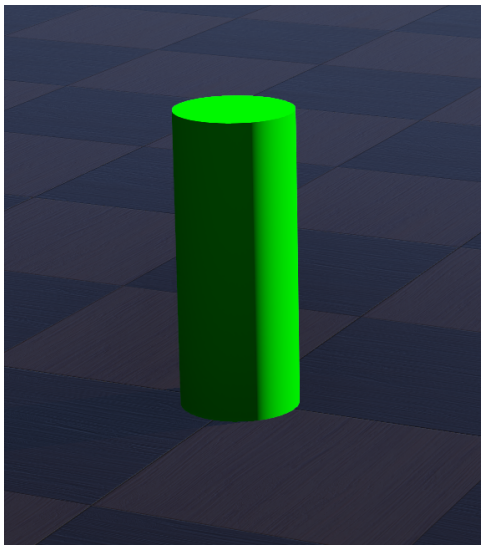

Figure 2: Green OOI to find.

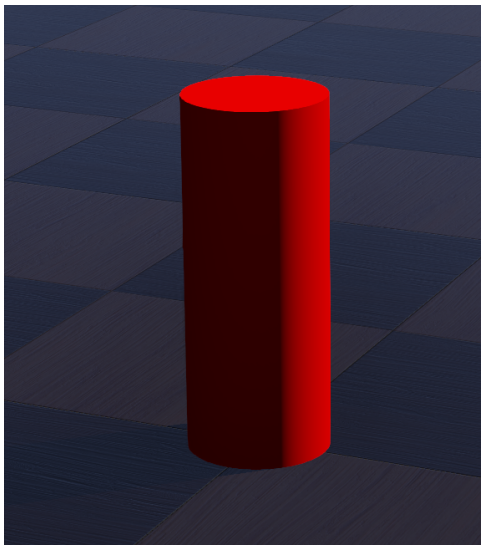

Figure 3: Red OOI

3.2 Advanced Task

In the advanced task of this assignment,you solution will be tested in a unknown world containing 4-6 red OOIs while still only having access to three robots. In this case, the robots will have to visit multiple OOIs sequentually. It is your responsibility to ensure robot-to-robot collisions do not occur, this could be done by using the provided webots distance sensor. Do not modify or add any sensors to the robots as those modifications will not be reflected in our testing world. If attempting this task, you must fill out the text file called `advanced.txt` and include it in your submission repository, If this is not done we will NOT run your solution on the advanced task world. Included in the document you must write 3-4 sentences detailing your approach and logic behind solving the advanced task.

4 Supplied code and file structure

A `BaseRobot` source code header file containing communication has been supplied and can be found in your private GitHub repository; details regarding the communication functions can be found in 5.6. Your file structure should follow the **exactly** one of the two layouts and naming structure detailed below (where `z1234567` is replaced with your own zID).

Note: two layouts are shown based on a discrepancy with the names elsewhere in the spec.

```
z1234567
├── advancedTask.txt (If attempting advanced task)
└── controllers
    ├── BaseRobot
    │   ├── BaseRobot.cpp
    │   └── BaseRobot.hpp
    ├── LeaderController
    │   ├── LeaderController.cpp
    │   ├── LeaderController.hpp
    │   ├── KeyboardConfig.txt
    │   └── Makefile
    └── ScoutController
        ├── ScoutController.cpp
        ├── ScoutController.hpp
        └── Makefile
```

```
z1234567
├── advancedTask.txt (If attempting advanced task)
└── controllers
    ├── BaseRobot
    │   ├── BaseRobot.cpp
    │   └── BaseRobot.hpp
    ├── LeaderController
    │   ├── LeaderController.cpp
    │   ├── LeaderRobot.cpp
    │   ├── LeaderRobot.hpp
    │   ├── KeyboardConfig.txt
    │   └── Makefile
    └── ScoutController
        ├── ScoutController.cpp
        ├── ScoutRobot.cpp
        ├── ScoutRobot.hpp
        └── Makefile
```

5 Detailed Requirements

5.1 Process

Below are the general tasks which must be completed in this assignment

- Leader Robot identifies all OOIs using LiDar point cloud data.
- Send Scout Robots to inspect all OOIS then Scout Robots report on the colour of the OOIs back to the Leader.
- Move the Leader Robot to within 1m of the green OOI.
- Leader Robot Logs all OOI locations and communications to a text file.

5.2 Robot IDs:

Each robot has a unique robot ID which is used for communication and logging purposes. The IDs are defined in the robot objects supplied in the world file. Do not edit or change these IDs as they are required for marking.

Robot ID:

- Leader - 0
- Scout 1 - 1
- Scout 2 - 2
- Scout 3 - 3

5.3 Base Robot

The BaseRobot is a class which is inherited by the Scout and Leader Robots. This is included as a means of sharing common code and concepts. Sensor parameters shared by both robots can be found in Table 1 (also see section 5.3.1). The functions outlined in Table 2 must be implemented and the variables within Table 3 must be used. You are free, however, to add additional functions and variables to the BaseRobot class.

Table 1: Base robot sensor parameters

Description	**Query ID**
GPS	"gps"
Compass	"compass"

Table 2: base robot functions

Method	Description
`BaseRobot()`	A constructor that initialises the robot, auto assign an ID and initialises the onboard sensors and actuators. Use the name of the robot inside to assign the automatic ID. See the docs for the method.
`~BaseRobot()`	Destructor
`virtual void run() = 0`	A pure virtual function. Subclasses will have to implement this member function (see subclass for further description).
`virtual void move(double speed) = 0`	A pure virtual function. Subclasses will have to implement this member function (see subclass for further description).
`virtual void rotate(double speed) = 0`	A pure virtual function. Subclasses will have to implement this member function (see subclass for further description).
`void keyboardControl();`	A function which allows a robot to be controlled by using WASD on a keyboard. The robot should only move when the respective key is pressed down.
`void updateCurrentPosition();`	Reads in GPS and Compass values and saves them to protected variables.
`void setTargetPosition(double x, double y);`	Saves the passed position to protected variables.
`bool moveToTarget(double stopDistance);`	When called the robot uses the stored target and current pose values then calculates what movements are required to reach the target. Stop distance sets the distance to the target position before stopping. If the robot is at the target 'reutrn true;' otherwise 'return false;'.
`void sendMessage(const std::string& toID, const std::string& data0, const std::string& data1)`	(PROVIDED) Used to send a message to another robot based on ID
`std::pair<std::string, std::string> receiveMessage()`	(PROVIDED) Used to receive a message from another robot, if the message ID matches the robot ID then a std::pair containing data0 and data1 is returned

Table 3: Required protected variables

Name	Description
`ID`	Stores the webots ID of the robot.
`currentPositionX`	Stores the last know robot x position.
`currentPositionY`	Stores the last know robot y position.
`currentYaw`	Stores the last know robot yaw rotation.
`targetPositionX`	Stores the destination x position.
`targetPositionY`	Stores the destination y position.

5.3.1 GPS and Compass

The GPS system offers accurate X and Y coordinates for tracking the robot's location, while the compass provides precise information about its yaw (z-axis) rotation. Detailed instructions on accessing these data sources are available in the following resources:

- GPS: https://cyberbotics.com/doc/reference/gps
- Compass: https://cyberbotics.com/doc/reference/compass

5.4 Leader Robot

The Leader Robot, as depicted in Figure 4, is equipped with a LiDar and is responsible for logging information to a text file and making swarm decisions. Webots parameters for the Leader Robot can be found in Table 4, and the required functions can be found in Table 5, however you are free to add additional functions and variables. You are free to design and implement your own data types and methods for identifying Object of Interests (OOIs). Similarly, how you assign OOIs to the Scout Robots is at your discretion.

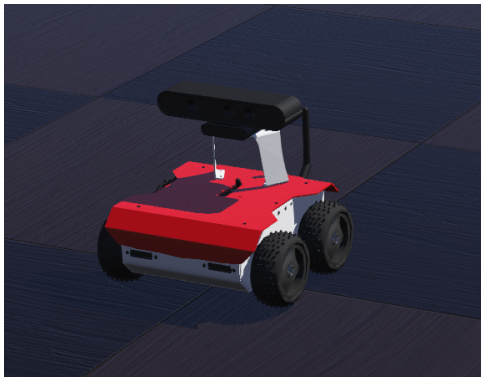

Figure 4: Leader Robot.

The tasks of the Leader Robot are to:

1. Scan the environment
2. Identify positions of OOIs
3. Log all positions of OOIs to text file
4. Send positions to Scout Robots
5. Log all communication and important events to an output text file
6. Receive location of green OOI

7. Move to with 1m of green OOI
8. Stops moving

The Leader Robot will always start in the same orientation but may be in a different x,y position requiring you to offset the LiDar data.

Table 4: Leader parameters

Description	**Query ID**
frontLeftMotor	"front left wheel motor"
frontRightMotor	"front right wheel motor"
rearLeftMotor	"rear left wheel motor"
rearRightMotor	"rear right wheel motor"
lidar	"lidar"

Table 5: Leader Robot functions

Method	**Description**
`LeaderRobot()`	A constructor that initializes the robot and initialises the onboard sensors and actuators..
`~LeaderRobot()`	Destructor
`virtual void run() override;`	A virtual function that runs the main loop of the controller
`virtual void move(double speed) override;`	A virtual function that moves the robot forward at a certain speed
`virtual void rotate(double speed) override;`	A virtual function that rotates the robot on the spot at a certain speed
`void scanLidarData()`	This function reads in the point cloud data and stores it as a private variable
`void fileOutput(const std::string& output)`	This function writes to the output file

5.4.1 Lidar

The specifications of the 2D LiDar used can be found below (see Table 6). Object of Interest (OOI) points can be grouped and extracted by examining the absolute positional difference between adjacent LiDar points. There is no x,y offset between the LiDar and Leader Robot gps.

- lidar: https://cyberbotics.com/doc/reference/lidar

Table 6: Lidar parameters

Parameter	**Value**
minRange	0.2m
maxRange	10m
field of view	360 degrees
Resolution	1000 points

5.4.2 Identification of OOI

One approach to grouping the position of each OOI is to loop through the LiDar data array and compare the distance between adjacent cartesian coordinates. If they are within a fixed distance of each other then they must be part of the same object. Every OOI will be at least 1 meter away from any other OOI. For this task, you may want to create custom structs or a similar datatype, in order to store information about the OOI and which robot is assigned to it.

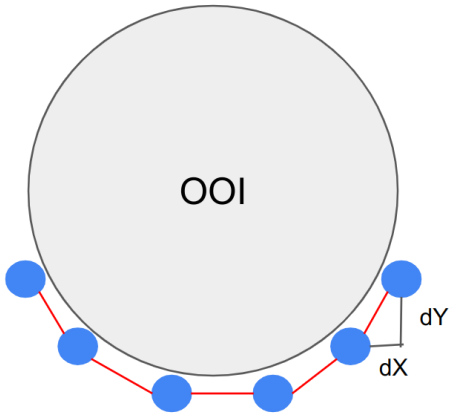

Figure 5: Example OOI with LiDar points.

5.4.3 Logging text file

For each condition detailed below, the output must be appended to the output file on a new line and printed to the terminal. Ensure your text file is called `output.txt`.

When an object of interest is identified:

`OOI discovered at x:<x value> y:<y value>`

When the Leader sends a position command to a Scout Robot:

`Target pose x:<x value> y:<y value> sent to robot <ID>`

When the Leader receives a colour message from a Scout Robot:

`Robot <ID> has identified a <colour> OOI`

When the green OOI has been found:

`Green OOI has been found, moving to x:<x value> y:<y value>`

When the Leader is within 1 m of the green OOI:

`Successfully arrived at the green OOI`

5.4.4 Keyboard Control

A '`keyboardConfig.txt`' file is required inside the LeaderController directory. In the Leader constructor, this file should be loaded in. The command '`keyboardControl=true`' will allow the user to command the Leader with keyboard controls, while '`keyboardControl=false`' will cause the program to operate autonomously. In keyboard control mode, the 'w' and 's' keys will move the robot forward and backward, respectively, while 'a' and 'd' keys will turn it on the spot left and right. The robot should only move if the respective key is pressed down.

5.5 Scout Robot

The Scout Robot shown in Figure 6 is equipped with a camera and is responsible for identifying and reporting the colour of the assigned OOI. Webots parameters for the Scout Robot can be found in Table 7 and the required functions can be found in Table 8, however you are free to add additional functions and variables. Scout Robots have a front facing distance sensor in addition to the supplied distances sensors on the e-puck robot, you are free to use any of these if wanted. While not required this can be used for obstacle avoidance and detection. You are free to create additional functions.

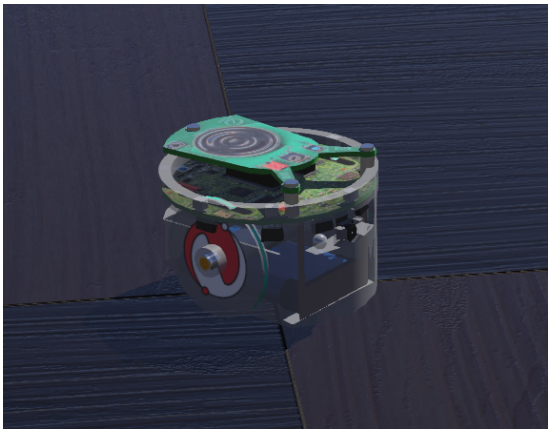

Figure 6: Scout Robot.

The task of the Scout Robot is to:

1. Receive target command from Leader Robot
2. Move to within 0.5m of the target
3. Identify OOI colour
4. Report to Leader Robot
5. Stops moving (unless instructed again, see Advanced Task)

Table 7: Scout parameters

Description	**Query ID**
leftMotor	“left wheel motor”
rightMotor	“right wheel motor”
camera	“camera”
distanceSensor	“distance”

Table 8: Scout Robot functions

Method	Description
`ScoutRobot()`	A constructor that initializes the robot and initializes the onboard sensors and actuators.
`~ScoutRobot()`	Destructor
`virtual void run() override;`	A virtual function that runs the main loop of the controller
`virtual void move(double speed) override;`	A virtual function that moves the robot forward at a certain speed
`virtual void rotate(double speed) override;`	A virtual function that rotates the robot on the spot at a certain speed
`bool readColour();`	This function identifies the colour of the OOI, It returns true if green OOI was found, otherwise false.

5.5.1 Camera Interface

The camera attached to the Scout Robot is used to identify the green Object of Interest (OOI). However, this assignment does not require computer vision; instead, it utilises the Webots recognition implementation to simplify operations.

To enable the camera recognition, use the following command:

```
camera->recognitionEnable(TIME_STEP);
```

To check the number of recognised objects in the camera frame, use the following command:

```
camera->getRecognitionNumberOfObjects();
```

In this context, a green OOI will return 1, while red OOIs will not register as recognised objects and return 0.

5.6 Communication Interface (Provided)

Within the provided base robot stub files, you'll find two communication functions. These functions offer a straightforward interface, allowing one robot to send two strings, data0 and data1, to another robot based on its unique ID. You can call the 'ReceiveMessage' function at each time step in your control loop. If a message with the robot's ID has been sent, 'ReceiveMessage' will extract the two strings and return them as a 'std::pair'. If there's no message, it will return a pair of empty strings.

The primary use of these functions is to enable the Leader Robot to transmit an x and y location to Scout Robots, while the Scout Robots can respond with their IDs and either "found" or "not found" status. It's important to note that this is a suggested use of the supplied interface functions, and you are free to use the data fields however you intend.

6 Compiling

All code must be able to be compiled in Webots 2023a runnning on Windows 11. If you don't use a windows device then you may develop your solution on your device, however you must verify it compiles on windows computer.

7 Testing

Three world files have been provided for testing although it is expected that you will modify it and test on your own worlds to address cases not covered in the supplied files. When marking we will test your solution on unseen world files which abide by all the restrictions specified.

7.1 Supplied Three Files

- 3 Targets - Leader Robot starting at (0,0)
- 3 Targets - Leader Robot starting not at (0,0)
- 4 Targets - Leader Robot starting not at (0,0) - Advanced Task

7.2 Robot Placement Restrictions

- Robot 0 will always start at an angle of 0, with its location satisfying the conditions: $x \leq 0$ and $-5 < y < 5$.
- Robot 1 will always start at coordinates (0.5, 0.5) with any rotation.
- Robot 2 will always start at coordinates (0.5, 0) with any rotation.
- Robot 3 will always start at coordinates (0.5, -0.5) with any rotation.

7.3 Restrictions and Notes

- A Scout Robot will never run into a static Leader Robot if travelling in a straight line to any OOI.
- No OOI will be occluded behind another OOI or hidden from the Leader Robot at world initalisation.
- Each OOI will have at least 1m separation between them
- It is up to you to move your robot to the desired position, however you do not need any path planning and trajectory following algorithms to complete this assignment.

8 Version Control

As part of this assignment, you will be **required** to use GitHub classroom. A repository has been established for all students and the starter code has been placed within this (it will NOT be released on Moodle). You can find this here (`https://classroom.github.com/a/FCVar_jj`) You will be expected to meaningfully use this version control platform as it is widely used in industry and provides a means of retrieving code in case of computer issues or getting old revisions of your work. Your submitted code on GitHub will be tested for plagiarism.

You must show regular commits to your private GitHub repository. This will be marked as part of the Solution Quality assessment. In this context regular commits are deemed to be commits showing a natural progression in development, such as completing a function or an aspect of the project.

To commit to your repository it is recommended that you use the GitHub desktop application. This can be downloaded here (`https://desktop.github.com/`). This provides a graphical interface to interact with the repository.

9 Assessment

This assignment is worth 35% of the total course mark. The allocation of marks for this assignment is described in Table 9.

Table 9: Mark Allocation for Assignment

Criteria	Weighting	Description
Keyboard Control	4%	If "keyboard=true" is set in "keyboard.txt" within the Leader controller class, the Leader controller can be controlled using the WASD keys. One mark is allocated for each key.
OOI Detection	6%	To meet this criterion, the program should correctly detect all three OOIs in a world with three OOIs and record them in the output text file.
Scouting Phase	6%	Scout Robots should navigate to within 0.5 meters of the target and report the associated colour.The Leader Robot should log all communications in "output.txt." This criterion involves using three OOIs.
Leader Interception	6%	The Leader should move to within 1 meter of the green target.
Advanced Task	5%	This criterion is evaluated based on the conditions outlined in the "Advanced Task" section.
Solution Quality	8%	The program ad hears to the function definitions and course style guide. Regular progress commits have been pushed to your private GitHub repository.

9.1 Submission

This assignment is due by 11:59 on Monday of Week 12 (27/11/23).

You must test your project on a computer running Windows (e.g., through VM, myAccess, or the Lab Computers) before submission. We can not guarantee marking on any other operating system. If you are developing on an apple or linux device then the software may not compile the same as on Windows. It is your responsibility to fix any incompatibility issues before submission.

You should zip your Webots project, name it as

`z1234567.zip`

where z1234567 is your zID, and submit it via Moodle. Your submission should contain all the files required to compile

and run your program.

9.2 Late Penalty

UNSW has a standard late submission penalty of: 5% per day, for all assessments where a penalty applies, capped at five days (120 hours) from the assessment deadline, after which a student cannot submit an assessment.

10 Plagiarism

If you are unclear about the definition of plagiarism, please refer to What is Plagiarism? — UNSW Current Students. You could 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). Please cite the source if you refer to open source code.

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