The University of Sydney

School of Aerospace, Mechanical and Mechatronic Engineering

Major Project

MTRX3760: Mechatronic Systems Design

Tutorial: Monday 12-3pm

Student ID:	500510331
Student ID:	500602739
Student ID:	500505234
Student ID:	490559677

Contents

1 Introduction

1.1 Project Background

With inspiration from the NSW Smart Sensing Network's (NSSN) recent media release, "*Floods, fires and defence: pushing the boundaries of smart sensing through situational awareness*". This press-release placed an emphasis on "storms, droughts, fires, and floods", with there being a call for **"new ways of managing resources and multi-disaster capabilities"**.

2 System Design

Our proposed solution for the NSSN utilises a TurtleBot3 'Burger', as shown in Figure 1, which has the ability autonomously navigate a terrain while addressing hazards as they are encountered. It also keeps track of the available hazard handling resources, and is able to navigate back to a base of operations when more are needed. We addresses the following scenarios:

- **Fires:** Navigating through an environment with various outbreaks of fire, putting them out (assumed ability) using fire suppressant .
- **Floods:** Navigating a water-filled environment, mitigating damage through placing sandbags (assumed ability) to disrupt water flow.
- **Storms** Navigating an affected environment, clearing debris (assumed ability) using onboard equipment.

As this is a singular robot that can be adjusted to handle different hazards, it achieves the goal of providing a solution with "multi-disaster capabilities", as well as effectively "managing resources". Our proof-of-concept also addresses the "storms, ... fires, and floods".

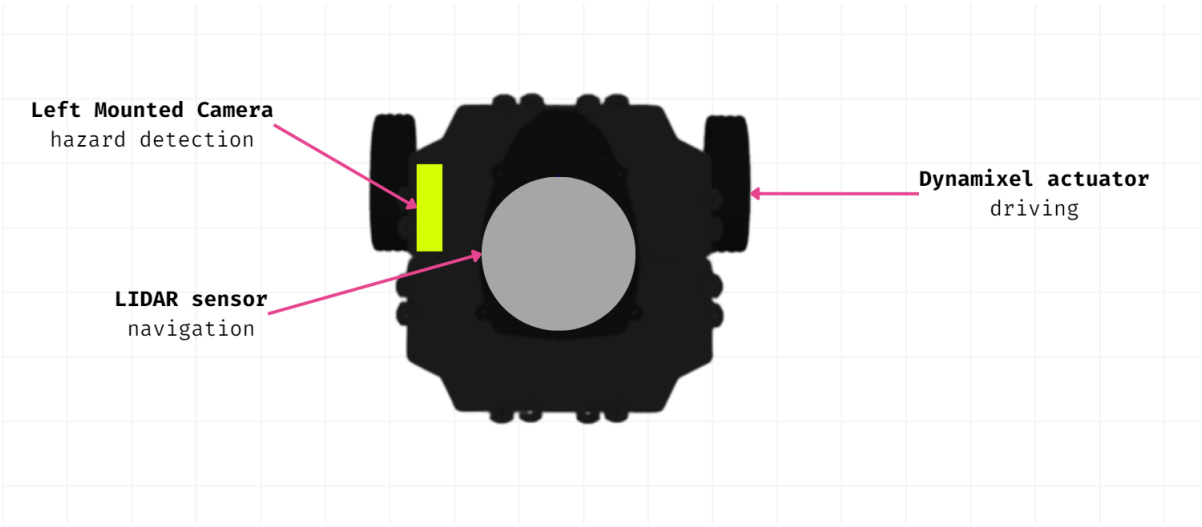

Figure 1: Turtlebot3 'Burger' Design

2.1 Key Outcomes

Pertaining to the proposed solution, the following functionalities were achieved:

- Wall following (left and right) to allow for autonomous navigation of an area with an established safety perimeter.
- Colour detection for hazard recognition.
- Checking each obstacle detected in front for whether it is or isn't a hazard
- Autonomously driving to a detected hazard to handle it.
- Coordinate determining for storing a safe location, as well as the last safe location before needing a restock.
- Quicker navigation to the last known safe location by not checking whether obstacles in front are hazards.
- Completion when the robot has gone through an area once, with no other hazards being detected.

This functionality was demonstrated in the presentation given on 30/10/2023. The attached linked will take you to this functionality demonstration:

`https://www.youtube.com/watch?v=2dPzNxlsU9E`

2.2 Modular Design

Our code is split into various classes, appropriately named after their respective functionalities.

CLidar: Handles the LIDAR data interpretation, and returns the distance from the robot to whatever is ahead, to the left, or to the right.

CCamera: Handles the camera data interpretation, with the input of a 'eHazardState' to determine what colour mask to detect.

CMotor: Handles the publishing of linear and angular velocity values to the robot's motor.

CCartographer: Handles the odometer data interpretation, and returns the robot's x, y, and w coordinates.

CWallFollower: Handles the logic for following a given wall, utilising data inputs from the LIDAR and odometer, and pushing out information to the motor.

CHazardHandler: Handles the logic for handling a detected hazard, utilising data inputs from the LIDAR and odometer, and pushing out information to the motor.

CHazards: Initialised a hazard object, with a certain quantity of resources required to handle it. This is utilised by the hazard handler

CBurger: Initisalises each component, as well as handles the logic for switching between the wall-following logic and hazard-handling logic.

eRobotStates: A header file which holds all states for hazards, hazard handling and wall following.

Through the principles of object-oriented design, as well as utilisng the taught c++ best-coding practices, this modularity ensures ease of code readability, and allows for easy adaptions to be made if functionality needs to be adjusted, or components such as the LIDAR or Camera need to change.

For instance, the NSSN's media release also mentions droughts, which our proof-of-concept currently does not handle. However, the modulatiry of our code would easily allow for this functionality. For an irrigation solution, a new state and appropriate colour mask could be easily added in eRobotStates.h and CCamera.cpp. CHazardHandler can then be adjusted with new hazard handling functionality such as watering plants, given the quantity determined by the CHazard object.

2.2.1 ROS Node Design Diagram

This diagram shows the "follow_the_wall" node and the interactions of the various components with the node including their communication channels.

As can be seen from the node diagram below, our design uses 1 communication node to subscribe to the information from the lidar, camera and motors. This subscription is held within each of the respective classes (CLidar, CCartographer, CCamera). The node also publishes information information to the motors (CMotor) in order to control the movement of the robot.

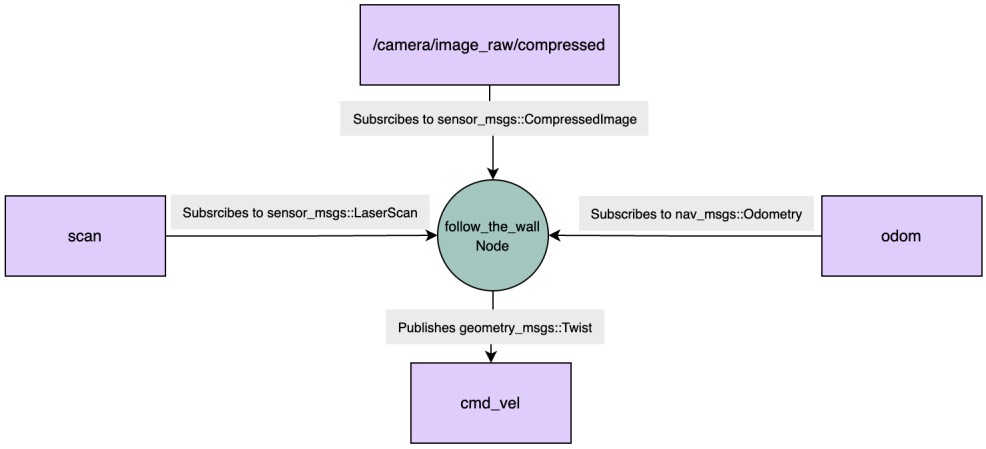

Figure 2: ROS Node Design Diagram

2.2.2 UML Diagram

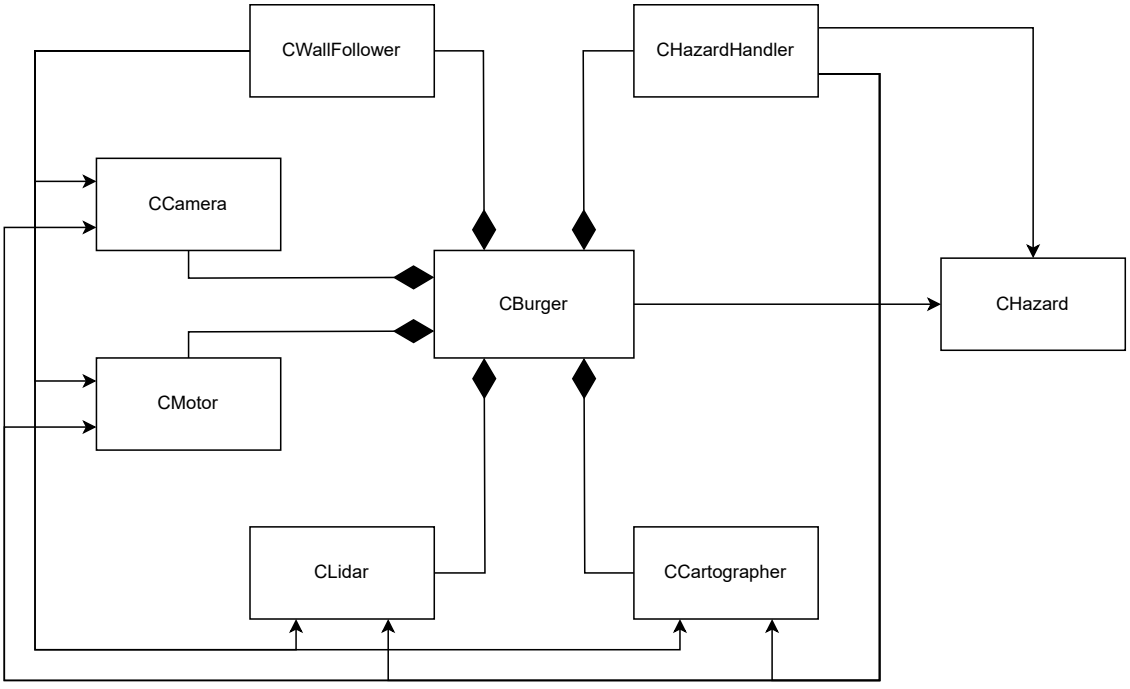

Figure 3: UML Diagram

2.3 Class Interaction Sequence

This class interaction sequence will cover the main functionality of the program, with this being the switching between the actions of wall following and hazard handling. This sequence will not include the class interactions of setting variables between the two classes that handle the aforementioned logic. An example of an interaction not included in this sequence is as follows, CBurger.RunNavigation calls mpHazardHandler -> SetStationX, passing it mpWallFollower -> GetStationX, if mpWallFollower -> GetStationX does not equal mpHazardHandler -> GetStationX

- Main calls the function RunNavigation() from CBurger
- CBurger.RunNavigation loops indefinitely while ros::ok() returns 1
- If mpHazardHandler -> GetResourcesCheck returns 1 and mpHazardHandler -> RestockCompleteCheck returns 0, CBurger.RunNavigation calls mpWallFollower -> WallFollowLogic, passing it mpHazardHandler -> GetWallFollowerFlag
    - mpWallFollower -> WallFollowLogic calls mpCartographer -> GetCurrentX, mpCartographer -> GetCurrentY and mpCartographer -> GetCurrentW, returning the current X,Y, and W coordinates
    - Based on the mpHazardHandler -> GetWallFollowerFlag input, mpWallFollower -> WallFollowLogic calls mpBurgerLidar -> GetCurrentRight, mpBurgerLidar -> GetCurrentLeft, and mpBurgerLidar -> GetCurrentFront, returning the current distances and setting them to mDistanceFollow, mDistanceOpposite, and mDistanceFront
    - mpWallFollower -> WallFollowLogic calls mpBurgerMotor -> BurgerControl, passing it through various member variable velocities for linear and angular velocity
- else if, mpHazardHandler -> GetResourcesCheck returns 1 and mpHazardHandler -> RestockCompleteCheck returns 0, CBurger.RunNavigation calls mpWallFollower -> WallFollowLogic, passing it mpHazardHandler -> GetWallFollowerFlag
    - Same class interactions as prior mpWallFollower -> WallFollowLogic
- else if, mpHazardHandler -> GetWallFollowlerFlag returns a value above 1 and mpHazardHandler -> RestockCompleteCheck returns 1, CBurger.RunNavigation calls mpHazardHandler -> HazardHandlerLogic passing it mpHazardHandler -> GetWallFollowerFlag, mpBurgerCamera -> ResetCameraFlag passing it mpHazardHandler -> GetHazardHandledFlag, and mpWallFollower -> SetCurrentState passing it the INIT state
    - mpHazardHandler -> HazardHandlerLogic calls mpCartographer -> GetCurrentX, mpCartographer -> GetCurrentY and mpCartographer -> GetCurrentW, returning the current X,Y, and W coordinates
    - Based on the mpHazardHandler -> GetWallFollowerFlag input, mpHazardHandler -> HazardHandlerLogic calls mpBurgerLidar -> GetCurrentRight, mpBurgerLidar -> GetCurrentLeft, and mpBurgerLidar -> GetCurrentFront, returning the current distances and setting them to mDistanceFollow, mDistanceOpposite, and mDistanceFront

    - mpHazardHandler -> HazardHandlerLogic calls mpBurgerMotor -> BurgerControl, passing it through various member variable velocities for linear and angular velocity

- else, CBurger.RunNavigation calls mpWallFollower -> WallFollowLogic, passing it mp-HazardHandler -> GetWallFollowerFlag

    - Same class interactions as prior mpWallFollower -> WallFollowLogic

3 Teamwork and Project Management

3.0.1 Teamwork

Name	Contribution
Samuel Kember	• Helped with final testing (only person with linux computer) • Helped with creating an instruction for bot setup • Helped with ensuring camera module was working • Helped with wall following logic in simulation • Helped with hardware testing and implementation • Helped with maze setup • Helped with code debugging and final editing • Helped with report and presentation
Josh Waldron	• Helped with final testing • Helped with LiDAR communication and setup • Helped with navigation logic • Helped with code debugging and final edits of the code • Helped with system reporting • Helped with hardware implementation
Leon Ortega	• Helped with final testing • Helped with simulation of system in hardware • Helped with motor control • Helped with LiDar operation • Helped with Algorithm for Open-CV object detection • Helped with Control logic • Helped with Turtlebot field testing • Helped with system reporting
Vaastav Varma	• Helped with final test • Helped with LiDAR communication and setup • Helped with maze setup • Helped with report and presentation

Table 1: Individual Contributions to the Project

3.0.2 Project Management

For the purpose of this project, we chose to use a ViModel design process approach to the probelm. This was an appropriate methodology as it allowed us to explore the problem, define requirements, and incorporate rigorous testing of each individual module as well as the final integrated system.

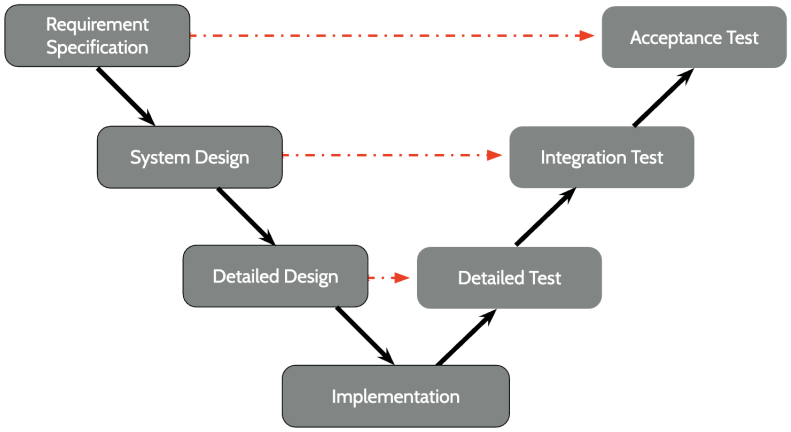

Figure 4: V Model Design Process

Requirement specification: The key requirements identified from the key design brief specifications were: Movement around the environment, detection of obstacles, tracking of resources used and identification of obstacles as hazards.

System Design: In the System Design, each functionality was implemented as a separate module ensuring that functionality can be added at a later date

Detailed Design and Testing: The detailed design involved deciding the different states of the Turtlebot based on the combined sensor inputs obtained while traversing the environment. All the subsystems were individually tested to ensure proper functioning.

Implementation: The code for the different modules was refactored and integrated for the overall system. The bugs encountered while these modules interacted with each other were fixed.

Integration Testing: In this stage, the workings of the Turtlebot with the refactored code was assessed. This stage was performed iteratively to achieve the desired internal interactions between the modules and the response of the Turtlebot with the environment. Some aspects of the system design were altered for smooth functioning and improving repeatability during testing.

Acceptance Testing: Once the integration testing was successful over multiple trials, the requirements for the system were revisited. Acceptance testing involved setting up the environment with different configurations of the hazards in the environment. Additionally,

the different types of hazards (red, blue, green obstacles) were tested. The Turtlebot was able to adapt to the changes in the environment and slight refinement in the boundary conditions of the different modules like the wall following logic enabled it to correctly navigate through these different test cases and satisfy the requirements.

3.0.3 Team communication:

GitHub Version Control: Different branches were used for the individual functionalities. This allowed us to use the same central code base without affecting the changes made by other team members.

Slack: The main communication medium for the team was Slack. We used this platform to share updates, schedule weekly meetings and keep a log of the team's progress.

3.0.4 Revision Control:

The following log is printed from the command `git log --pretty=format:"%h%x09%an %ad %s" --date=short`

```
3f8a2e8d Sam Kember 2023-10-29 Fully working - demo video is ready to be recorded.

65ad78e Sam Kember 2023-10-29 Group effort in optimising code to operate in maze.  Commit to save progress.

8a36ba4 Leon Ortega 2023-10-29 Logic for waiting at home and supply replenishing has been adjusted.

7b192c8 Sam Kember 2023-10-29 Navigates maze to detect first object, however, when it returns home it does not stay stationary to mimic replenishing of supplies - also does not replenish values of supplies in software.

2c3af39 Sam Kember 2023-10-28 Successfully finds wall after hazard detection.  Ready to test in full-sized maze with the whole group.

4e7f29b Leon Ortega 2023-10-28 Hazardhandler has been thoroughly worked through, previous commit issue should be working.

4c3af39 Sam Kember 2023-10-28 After obstacle, successfully finds wall but does not engage in wall follwing.  Minor changes made but needs further debugging.

1af3e60 Leon Ortega 2023-10-28 Changes to obstacle detection range, looks to run fine in simulation.  Ready for Sam to test.
```

5b1fe2a Sam Kember 2023-10-28 More accurate in finding hazard. But after obstacle, successfully finds wall but does not engage in wall following. Issue in finding wall after detecting hazard.

63d2a83 Josh Waldron 2023-10-28 Small changes in hazardhandler logic done. Ready for Sam to test.

8b3d1a6 Sam Kember 2023-10-28 Can now go around corners properly. Colour detection works perfectly. Issue in consistently detecting hazard when wall-following.

6f9a4e7 Sam Kember 2023-10-27 Wall following is failing to go around corners, corners are taken too sharp.

4ba8d32 Sam Kember 2023-10-27 Began testing in hardware, some changes have been made to hazard and wall following to update numbers that were different because of the hardware.

2dc28f0 Leon Ortega 2023-10-27 Wall following logic now working in simulation, merged with main brannch.

5dc28f0 Leon Ortega 2023-10-26 Wall following logic sort of working, however, in simulation doesn’t run well

9db35f1 Sam Kember 2023-10-26 Wall following logic better however still is not able to deal with all case logic well.

2dcdb51 Sam Kember 2023-10-25 Wall following logic sort of working, however simulation doesn’t run well

9ced1fe Sam Kember 2023-10-25 New branch for the wall following logic.

01c2d22 Leon Ortega 2023-10-25 Progress made on the hazard detection, working in the lab with Josh in simulation. The hazard dectection module is ready to be tested. Branch merged

33a3f00 Josh Waldron 2023-10-25 Made a new branch to work on hazard detection.

be198c6 Vaastav Varma 2023-10-23 Left wall following is now working in simulation ready to be tested on the actual turtlebot. The branch has been merged with the main.

104be4f Leon Ortega 2023-10-22 Lidar and motor working, have merged with main.

85b1cc3 Leon Ortega 2023-10-22 Made a new branch to check Lidar and motor from last assignment.

718b4ce Vaastav Varma 2023-10-22 Working on navigation issues, need to use left wall following. Have made a new branch

288ec3d Sam Kember 2023-10-20 Figured out this issue with the camera, now using the compressed image as opposed to the main raw one.

94a62b4 Sam Kember 2023-10-20 Working on camera module (in a new branch), issues with connectivity on a new computer

973e6b0 Leon Ortega 2023-10-19 Add files from Project 1 as a starting base

49593b5 Sam Kember 2023-10-18 Updated Connecting to Bot Instructions

2842aee Sam Kember 2023-10-18 Created Connecting to Bot Instructions

bd873cc Sam Kember 2023-10-18 Initial commit

4 Google Drive link for code

This link will take you to the .zip file containing the code written by this group for the assignment

https://drive.google.com/file/d/1G1DTDdFNR7sL24MWny41h8b6_YyHEd1p/view?usp=sharing

5 Self Assessment

5.1 Presentation

5.1.1 Introduction / Motivation

We had an informative introduction to the problem that our project aimed to solve and clear motivation behind our groups project. ⇒ **5/5**

5.1.2 Design Overview

We gave a clear design overview of the project using a Node diagram and a class diagram (made for a presentation so not quite a UML). However, the presentation could have better demonstrated the design of our project using clear visualisation ⇒ **10/15**

5.1.3 Functionality

The video demonstrating our functionality showed both the user view (robot going around the environment) and the information given to the operating (display of the computer screen). Our functionality correctly addressed our design specifications in line with the design brief in an interesting and creative way. The final functionality was fully operation and we were able to showcase all functionality planned. Given this, i have given full marks for functionality (Bonus marks are no assumed to be counted in the self-assessment section) ⇒ **15/15**

5.1.4 Teamwork & Project Management

We clearly shows how our team was able to work together using Github and slack and the project management method used in the project. ⇒ **5/5**

5.1.5 Presentation, organization, clarity

The presentation was professional and clearly showed our design, functionality and method of working together. ⇒ **5/5**

5.2 Report & Code

5.2.1 System design

The system design has been well documented using a class interaction sequence, ROS NODE Diagram and UML Diagram. The system design focused on the modulation of code. While we believe we were relatively successful at this, improvements could be made in the Wall follower and Hazard handler classes. ⇒ **15/20**

5.2.2 Teamwork Project Management

The report clearly shows the individual contributions of each team member, the git log of contributions to the GitHub and the communications methods used. The project management style used (V-model) is also explained in the context of the project. ⇒ **10/10**

5.2.3 Code

The code is well commented on with constant naming conventions throughout. However in this section, there is always room for improvement and therefore we will mark ourselves down slightly. ⇒ **15/20**

5.3 Hardware Return

All hardware used was returned to our Tutor at the completion of our presentation ⇒ **5/5**

5.4 Total

Altogether, we think this submission would score an 85. While the functionality was perfect, the code quality and design could be improved upon. ⇒ **85/100**

6 Code Appendix