ASSIGNMENT 2 - Specification

MTRN3500 Computing Applications in Mechatronic Systems - 2023

Alexander Cunio & Jay Katupitiya

Change log:

- **09/10/23**: Original release

1 Assignment Aims

A systems integration practical project with an Unmanned Ground Vehicle (UGV)

- To implement object-oriented programs using C++.
- To create a multithreaded program for concurrent activity.
- To demonstrate inter-thread communication and thread management.
- To interface with live streams of data from industrial sensors and to use them to assist in the tele-operation of a UGV.
- To interface with existing software to leverage pre-existing functionality.

Note: The word “module” is used to describe a piece of self-contained software. The word “thread” is used to describe a module that is running.

2 Background

Often, in writing software for Mechatronic Systems, we need to integrate software modules that interact with sensors and actuators and software modules that carry out computational tasks, under the supervision of a thread management software module. Rather than building a monolithic application, a multi-threaded approach can be used to separate functional tasks and to simplify the software development, upgrade and maintenance. Such an approach, alongside appropriate exception handling, has the added benefit of limiting the impact of software errors that may occur during operation, via thread isolation (i.e., if there is an error in one subsystem the rest are able to continue running normally). As such, we need to use inter-thread communication to allow for these specialised threads to communicate with each other.

3 What is Required of You

In this assignment, you are expected to write multiple independent modules that interact with each other through shared memory (variables that are shared between independent threads in a program). Your development should include the following software modules:

- a Thread Management Module to set up shared memory and to startup, monitor and shutdown the operation of all the modules listed below.
- a Laser Module to interface to a data stream originating from an LMS151 laser rangefinder and store the data appropriately for access by other threads.
- a GNSS Module to interface to a data stream originating from a Novatel SMART-VI GNSS receiver and store the data appropriately for access by other threads.
- a Controller Module to read signals coming from an x-box controller and store the data appropriately for access by other threads.
- a Vehicle Control Module to direct the controller signals to drive and steer the unmanned ground vehicle.
- a Display Module to graphically display the x-y data of the laser range finder and GNSS data in a virtual world.

You are then required to (i). demonstrate receipt of Laser and GNSS data, (ii). demonstrate steering and propulsion control of the UGV and (iii). demonstrate the graphical display of laser and GNSS data. See from Section 4 onwards for the details. Start by completing Part I in Section 4 and then follow the other parts.

4 Part 1 - Thread Management Module

The purpose of the thread management module is to set up shared memory, start up all threads and monitor the operational health of all the threads and finally to shutdown all the threads. Develop the thread management module to:

1. Set up shared memory. The shared memory module must have the capability to:
   (a) Provide read/write access to Thread Management data.
   (b) Provide read/write access to Laser data.
   (c) Provide read/write access to GNSS data.
   (d) Provide read/write access to Vehicle Control data.
2. Start all other threads one by one in the most logical sequence suitable for tele-operation of the UGV.
3. Monitor the heartbeats of all the other threads.
4. Carry out a complete shutdown in the event the thread management thread detects the failure of a critically important thread.
5. Attempt to restart a failed non-critical thread.
6. Carry out a routine shutdown of all the threads in response to pressing the ‘q’ key (within the thread management terminal window).
7. Indicate to all the other modules that it (the thread management module) is alive and operational. If thread management fails all other threads should terminate.

Thread Management is the most important module and therefore will form a critical threads.

5 Part 2 - Laser Module

The UGV mentioned earlier will have its own server running onboard. The purpose of the server is to allow you to interact with the UGV. Therefore, in this part your first task is to establish a Wi-Fi connection with this server over the wireless network (you should be able to do this from either your laptop or the lab computers). Use 192.168.1.200 as the IP address and 23000 as the port number to connect to and receive laser data (further information in service details document). After authentication, you will be allowed to connect to the laser rangefinder to enable you to collect laser range data. The most recent laser range data you collected must then be stored as an array of X, Y coordinates in the shared memory for other modules to use (an appropriate struct to hold the data is available within `smstructs.h`).

Incorporate mechanisms in the laser module to update its own heartbeat so that this is observable by the thread management module and for the laser threads to respond to shutdown commands form the thread management module.

The laser module will form the primary input for the UGV's tele-operation and is thus a critical thread.

6 Part 3 - GNSS Module

Similar to the Laser, the GNSS runs on a server on the weeder thus requiring a Wi-Fi connection to the robot. The device will run at port 24000 to communicate via a TCP connection to 192.168.1.200. The server does not need any authentication. As soon as you connect to the GNSS receiver it will allow you to get global position data records as a continuous stream of binary data. The arrangement of this binary data can be inspected within the GNSS manual extracts document. Process the binary data to obtain the X, Y position in UTM coordinates and height in meters, and make them available in the shared memory for other modules to use. In addition, this module should print on the screen, Northing, Easting and Height in meters + the CRC value (a value that indicates data integrity).

Incorporate mechanisms in the GNSS module to enable the thread management module to detect the heartbeat of the GNSS thread and for the GNSS thread to respond to shutdown commands form the thread management thread.

The GNSS module provides secondary input information to the driver who can continue without this and is thus a non-critical thread.

7 Part 4 - Controller Module

You will use an X-box controller to provide control input to the tele-operation system. The speed of the robot will be controlled by the right and left triggers (right for forwards motion and left for reverse). The steering should be controlled by the x-axis of the right joystick. You will be provided with a controller for testing within the laboratory sessions. Make the speed and steering commands generated this way available to the vehicle control module through shared memory. In addition, program one of the X-box buttons to indicate to the thread management thread to shutdown all modules at the end of your tele-operation.

For you to more easily interface with the controller, an interface class has been written. The available data structures and functions are available in the `controllerInterface.h` header file. You should not change this file at all. Note, it is a C++ class (not CLR) when you include it in your project. This will provide you a means of connecting to a controller, getting the state of the controller, checking

connection, and other useful functionality. At the top there is a `#define` defining the type of control input so you can change it between controller input and keyboard input. This can be used so that you are able to test your code at home where you might not have an X-box controller available.

Incorporate mechanisms in the X-box module to enable the thread management thread to detect the heartbeat of the X-box thread and for the X-box thread to respond to shutdown commands form the thread management thread.

Note: If X-box is disconnected or turned off, set the speed and steering to zero!

This forms a critical thread for tele-operation.

8 Part 5 - Vehicle Control Module

The purpose of the vehicle control module is to send control commands to the UGV. As this is running on a server on the UGV, first establish a Wi-Fi connection with the onboard server to communicate via port 25000 with a TCP connection to 192.168.1.200. After authentication, you must send the UGV commands in the form of the following ASCII string to make it move.

```
# <steer> <speed> <flag> #
```

Replace the field `<steer>` by a numerical value between $\pm 40^{\circ}$ as the required steering angle and, `<speed>` by a numerical value between $\pm$1m/s as the required speed. These values should be based on your control input coming from the controller module (values available via shared memory). Replace the `<flag>` field by 0 and 1 alternatingly to indicate you are actively controlling the vehicle (it should alternate in consecutive messages). The spaces and the # are required.

Incorporate mechanisms in the vehicle control module to enable the thread management thread to detect the heartbeat of the vehicle control thread and for the vehicle control thread to respond to shutdown commands form the thread management thread.

The vehicle control module is important in tele-operation thus forms a critical thread.

9 Part 6 - Display Module

The display module is a means of providing a useful way to display collected sensor information to the user. Particularly, it should be able to portray the Laser data appropriately (representing the current data). All data should be plotted relative the the vehicle base frame. To complete this task Matlab will be utilised as a display engine. As such, there will be two components: the module within your Visual Studio codebase and the Matlab software. The latter is provided for you and you do not have to write this (see provided Matlab files with starter code on Moodle which can be run directly).

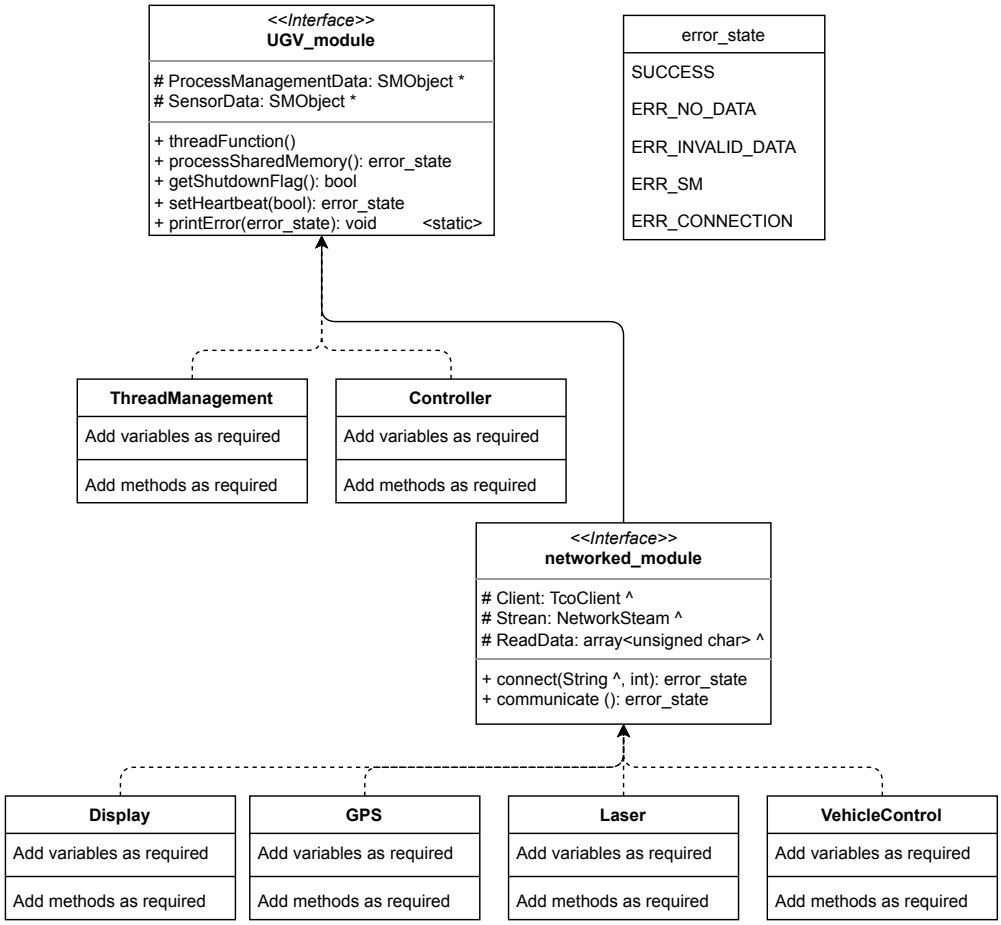
To interface with the existing Matlab display engine, data must be published to a local TCP/IP connection. It will read this data continuously and plot it appropriately. The format of the message sent should be an array of raw Byte data, sending the Cartesian co-ordinates of the data points. This should have enough size to fit 361 laser data points. A code snippet has been provided with the Appendix to perform this task of sending the data.

Similar to other modules, the display module will require access to the shared memory to collect data, modify heartbeats appropriately, and respond to shutdown commands.

The display module provides all the collected information to the user as a first person view, who would otherwise be driving blind, thus forms a critical thread.

10 Software Architecture

As this software system should be written as object oriented code, there is a specific layout of inheritance that should be followed. This is provided in the attached diagram, describing the class hierarchy. Particularly, abstract base classes are provided and each module must appropriately inherit from these and implement the required functions. For further information on working with classes and inheritance review the *week 1 lecture*.



11 Software Setup

The software for this system will exist within a single *project* within Visual Studio 2022. This means there will be a single *main* function that launches the first thread (Thread Management module) and from here all others are created.

Setup of the project will be conducted in a similar way to assignment 1, except for differing project settings. Review the assignment 1 setup video if you require more information of the use of Visual Studio. This section will summarise the settings that must be entered into your project, assuming download of the supplied starter code from Moodle which should be placed within the *solution directory* (same as assignment 1). Additionally, it will outline all files that you must create and complete.

General project settings:

- Configuration Properties->General->Output Directory: *$(SolutionDir)executables*
- Configuration Properties->General->Target Name: *$(ProjectName)_{zid}* – where zid is your Zid
- Configuration Properties->Advanced->Character Set: Use Unicode Character Set
- C++->General->Additional Include Directories: *$(SolutionDir)include*
- Linker->General->Additional Library Directories: *$(SolutionDir)libs*

The following describes the library files that should be included within the 'Linker->Input->Additional Dependencies' setting field for each provided configuration and platform.

Configuration	Platform	Library file
Debug	x64	ControllerInterface_x64_Debug.lib
Release	x64	ControllerInterface_x64_Release.lib
Debug	Win32	ControllerInterface_Win32_Debug.lib
Release	Win32	ControllerInterface_Win32_Release.lib

Project files:

For each module, you will need two additional files that will include the implementation of said module. These will be a header file that will define the class structure and a cpp file that will implement all class functions.

Adding additional files can be done in the solution explorer as below:

- For header files this will be: right click on Header Files, select add->new item, create a header file of appropriate name and type, click add.
- For cpp files this will be: right click on Source Files, select add->new item, create a cpp file of appropriate name and type, click add.

The following table summarises the names that the required files should take. These should be used **EXACTLY** in your project.

Module	Header	Implementation
Thread Management	TMM.h	TMM.cpp
Laser	Laser.h	Laser.cpp
GNSS	GNSS.h	GNSS.cpp
Controller	Controller.h	Controller.cpp
Vehicle Control	VC.h	VC.cpp
Display	Display.h	Display.cpp

In addition to these, you must also include a main file for your project that will include the overall *main* function. This should be called *Main.cpp*.

Note: To help you get started a template TMM.h file has also been provided in the starter code compressed folder.

12 Using the simulator

As with the last assignment, you have been provided with a simulator to test your code when outside the lab. It can be installed with the installer provided on Moodle and run in the same way to last assignment's simulator (extract the files and double click on the executable).

To change between running code on the robot and the simulator requires one minor change. The only difference is the IP address it is running at (all ports and other connection information, including authentication) should be the same. In the `networked_module.h` header file, you will need to change `WEEDER_ADDRESS` to `127.0.0.1` when running on the simulator and `192.168.1.200` when running on the actual UGV (connect to via Wi-Fi in the lab).

If you believe you have found any bugs in it please let us know to review.

13 What to look at next?

Now that you have read the assignment specification (do not forget to read submission instructions below), you can start setting up your project based on the instructions. There is one additional document (found on Moodle) that you should review. This is the service details document containing important information for connecting to sensors and actuators.

Additionally, you should review all the starter code provided, particularly all the header files in the *include* folder. Note that, there are four files that you will need to interact with directly (smstructs.h, UGV_module.h, networked_module.h, and controllerInterface.h).

14 Software Considerations

When writing your software you must remember to be following good software development practices. Particularly, in this assignment you must be:

- Following the prescribed software architecture throughout all modules as outlined in the previous sections.
- Undertaking appropriate exception handling to ensure safe program execution
- Performing data validation when reading from sensors (Laser and GNSS)

15 Git Usage

As part of this assignment, you should be employing Git to safely store and version manage your software. Similar to last assignment, a repository has been established for you within GitHub classroom. Please follow this link to access the repository. You MUST use the repository set up for you in GitHub classroom.

16 Assessment

Important note: None of the marks will be valid if plagiarism is detected in the submitted code or if you did not submit your entire source code by the deadline.

1. You are required to write all the software to ensure the operation of the above-mentioned threads in your own time. You can either demonstrate functionality on your own computers or the lab computers connected to the Weeder.

2. Complete **progress check** in week 8. The first progress check is to demonstrate the skeletal layout of all the modules and some base functionality and the operation of the inter-thread communication through heartbeat management. The demonstration required is,

   (a) Showing the completed class structure for ALL modules. The functions do not have to be implemented (2 mark)

   (b) The Laser thread should be printing live laser range data on the screen (the data must be presented as the 361 concurrent data points converted to Cartesian coordinates). (2 marks)

   (c) The automatic shutdown of all other modules in response to a routine shutdown of the thread management thread. A routine shutdown of the thread management thread is "pressing the 'q' key" to exit its main loop. (2 marks)

   You must get this marked during your allocated lab time in week 8. Late penalties per School policy applies.

3. Complete the final **assessment** in week 11 (early submission available as per below). This is an assessment and therefore there are no repeat opportunities for these tasks. You get only one opportunity to demonstrate.

   (a) Plotting laser data on the graphical display. (3 marks)

   (b) The operation of the GNSS thread printing live GNSS data to the terminal in a meaningful manner (this includes Northings, Eastings, height, and CRC). (3 marks)

   (c) Demonstrating steering and propulsion of the vehicle in response to joystick commands. (2 marks)

   (d) Demonstrating shutdown of all threads in response to a keyboard command. (2 marks)

   (e) Answering the two questions your demonstrator will ask about your software. (3 marks)

   (f) Following prescribed software architecture (2 marks)

   (g) Error handling and data validation throughout modules (4 marks)

   Note: You will not be awarded any marks if your software deviates significantly from the prescribed architecture.

4. **DO NOT FORGET** to submit all your software in a zip file named z1234567.zip with 1234567 replaced by your student number. This zip folder should directly contain all the .h and .cpp files you created for each module (twp per module plus the main file). Submit your file by 11.59 pm of 24 November 2023. Failing to do this will apply late submission penalties to your full assignment marks.

17 Early Submission and Bonus Marks

You will be provided with the opportunity to earn bonus marks for early submission of your assignment. This only applies to the final assessment submission and NOT the progress check.

Submitting one week early (practical assessment in your week 10 lab class and final code submission by 11.59 pm of Friday of week 10) will grant one bonus mark. Submitting two weeks early (practical assessment in your week 9 lab class and final code submission by 11.59 pm of Friday of week 9) will grant two bonus marks.

18 Penalties

1. If you are found to have plagiarised you will get zero marks for the assignment and you will be reported to School and University authorities – a plagiarism check will be run on your final assignment submission.
2. Late penalty will be applied to this assignment according to school policy. Please refer to the late penalty details in the course outline. Any submission made later than five calendar days will get zero marks.
3. If you do not attend the assessment lab session you must apply for special considerations through myUNSW.

Appendix

The blow function can be used within your display module for sending data to the Matlab display engine.

```
void sendDisplayData(array<double>^ xData, array<double>^ yData,
        NetworkStream^ stream) {
    // Serialize the data arrays to a byte array
    //(format required for sending)
    array<Byte>^ dataX =
        gcnew array<Byte>(SM_Laser_->x->Length * sizeof(double));
    Buffer::BlockCopy(SM_Laser_->x, 0, dataX, 0, dataX->Length);

    array<Byte>^ dataY =
        gcnew array<Byte>(SM_Laser_->y->Length * sizeof(double));
    Buffer::BlockCopy(SM_Laser_->y, 0, dataY, 0, dataY->Length);

    // Send byte data over connection
    Stream->Write(dataX, 0, dataX->Length);
    Thread::Sleep(10);
    Stream->Write(dataY, 0, dataY->Length);
}
```