Assignment 2: Threading, Synchronisation and Data Integrity

Revision Date	**Change**
15-SEP-2021	Initial Release

Table of Contents

General Information

Intent: Skills in utilising, classes, abstraction, data structures, threading, data synchronisation and documentation will be assessed.

Individual Task

Weight: 20%

Task: Write a program in C++ using object-oriented paradigms that shares data originating from a range of Mechatronics sensors between a number of threads. Ensure data integrity between threads and enable relating data between them via suitable data structure that enables time synchronisation and subsequently interpolation of data for task at hand. Supply appropriate auto-generated documentation utilising inline source mark-up.

Rationale: In a Mechatronics System, sensors produce data at varying rates. Decisions need to be made based on correctly associated data in near real-time. Threading and synchronisation are ways to ensure the system performs as intended, with guarantees on the responsiveness of the system to incoming data changes, processing constraints and system behaviour.

Your task is to (a) create a number of threads that can handle buffering incoming data (b) create separate thread(s) to process the data from the sensors that ensures synchronisation of the data between producers / consumers and produces an output by extrapolation of data (extrapolation is the process of estimating, beyond the original observation) (c) implement a graph, and search the graph for optimal paths, incorporating information about the platform motion into the graph.

Due: As per subject outline

Specifics

An aircraft is patrolling space surrounding a base station. The aircrafts task is to localise enemy aircraft (bogies) that enters the airspace and then intercept the bogie (get within 200m of it).

There are two modes of the assignment

1. BASIC MODE has stationary bogies
2. ADVANCED MODE has moving bogies, they move at a constant speed through the airspace. If they leave the airspace they turnaround and come through again.

Students attempting ADVANCED MODE also need to have completed the BASIC MODE.

Aircraft

Your aircraft (friendly) is controlled by supplying desired linear velocity and angular velocity (V and ω) that the on-board systems immediately respond to, as long as these controls are within operational parameters of aircraft.

The controlled linear velocity cannot be less than a terminal velocity V_TERM = 50 m/s, otherwise the aircraft will stall and fall from the sky, or above max velocity V_MAX = 1000 m/s. The combination of linear and angular velocity cannot exceed G_MAX = 6G (safety limit for the pilot : calculated as V * ω / g ; g being gravitational force).

A watchdog timer in the control systems monitors that control input is supplied every 25-50ms. If the control is not supplied every 50ms the control system will fail (and the application is terminated). If the control is supplied more frequently than 25ms it will be ignored.

Your aircraft can not leave the designated airspace (AIRSPACE_SIZE is the total airspace), you are required to remain around the base and defend it.

Sensor Data

The aircraft is equipment with:

- high-precision system provides the aircraft precise pose: x,y and θ (instantaneous non blocking call)
- directional radar that provides range and bearing to the bogies (from the aircraft) updating at FRIENDLY_REF_RATE (blocking call)

Apart from the on-board Radar the aircraft can:

- Receive readings from a radar situated at the base station that provides range and velocity to bogies (from the base station) at BSTATION_REF_RATE, base station is at location x= 0 y=0 (blocking call)

Given the aircraft received measurements from onboard radar, yet needs to chase bogies around the base, measurements need to be transformed into a “global” reference frame. This allows to achieve control as well as estimating velocity of bogies (in ADVANCED MODE).

Path Planning

You will need to consider the aircraft control (combination of turning and going straight) and the bogie position (in ADVANCED MODE also the bogie velocity) to be able to intercept aircraft efficiently. A graph search is the best way to handle computing paths when not able to derive equations analytically to solve a problem.

In BASIC MODE the graph would have bogies as nodes and have a cost to reach the nodes along the edges of the graph. The search in the graph would then provide the optimal order to visit the nodes.

In ADVACED MODE visiting a node (intercepting an aircraft that moves) would result in a different series of actions to intercept the remaining bogies. Therefore, this becomes a combinatorial search where committing to a certain order of intercepting bogies.

Aircraft Control

To achieve intercepting we also recommend to use a closed loop controller that will adjust the aircraft velocity and turn rate as your chasing the bogie. This is also how real-world systems work as it is incredibly hard to maintain a straight line or align yaw of your vehicle. Consider that even a 2deg error in orientation will result in 175m error over 5km (and your flying over a larger area). Your control could be a P controller or a method know as pure-pursuit which is described in a separate document and adjusts velocity and angular velocity as you fly through airspace using a lookahead point.

Simulator Library

A library (libsimulator.so) and header files encompassing the simulator is provided. To illustrate how to use the simulator a sample project containing: CMakeLists.txt and simple main.cpp exploiting the simulator is **available under student git repository (skeleton/a2_skeleton)**. You will need to change this code, in your submission the main should not have any functions, rather the sole role of the main should be instantiate objects, start threads in objects and connect objects.

The complete API for the Simulator is **available under your git repository (skeleton/a2_simulator_doc).** To access the API specifics, open **index.html** within above noted folder (for instance “firefox index.html”). Peruse Class List and view the public methods available under Simulator class. You can also see this information in the corresponding header files.

Task

Create an instance of the simulation and a number of threads to achieve control of an aircraft with an aim of (1) estimating the position of an enemy aircraft (bogies) over time and (2) continuously intercepting these bogies (getting within 200m of the bogie).

Your system will need to

1. Obtain data from the aircraft and/or the ground station
2. Utilise the data to estimate the position of the bogie
3. Control the aircraft within the watchdog imposed limits and aircraft specification: V_TERM - terminal velocity, V_MAX - max velocity, G_MAX - max G profile.
4. Stay within the airspace around the base station (total area is AIRSPACE_SIZE x AIRSPACE_SIZE with base station in middle).
5. Intercept as many aircraft as possible

Basic task (GameMode::BASIC) – Bogies do not move

1. Estimate controls action that will lead to your aircraft intercepting the bogie, execute the control actions
2. When all bogies are intercepted new bogies are respawn

Advanced task (GameMode::ADVANCED) – Bogies have linear velocity when going through airspace and then turn outside of airspace

1. In order to intercept, linearly extrapolate the bogie position in time (predicting where the bogie will be in future will allow you direct your aircraft)
2. Determine the control actions and execute them
3. If bogies leave the airspace they turn around and head back into airspace
4. When all bogies are intercepted new bogies are respawn

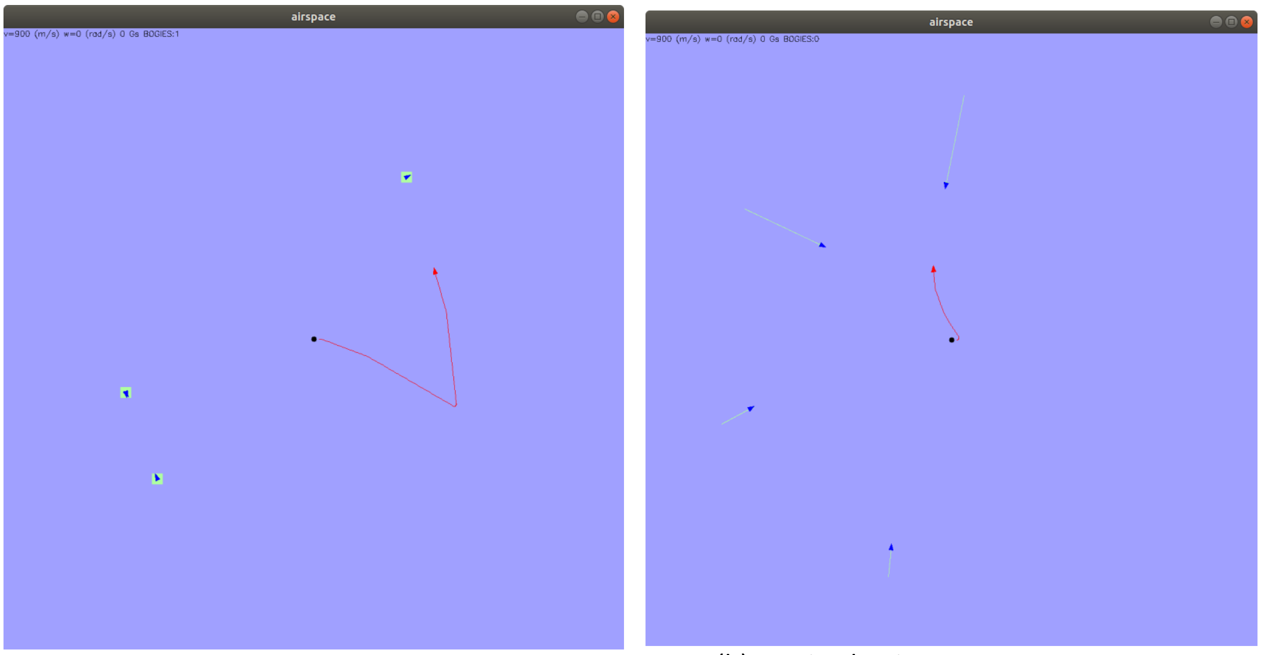
Therefore, you will need to design classes mapped to the task and functionality (refer assessment criteria).

Create a Main that

1. Creates objects from your classes, the classes should have a suitable number of threads for the task
2. Handle concurrency using appropriate data protection
3. Achieves goals for the task
4. Handles BASIC MODE / ADVANCED MODE (via passing a parameter on command line to the main

Supply unit tests that:

1. Calculate the bogie position correctly in global space (with reference to base station) – posetest.cpp
2. Associate the readings from aircraft and base station correctly to bogies – associationtest.cpp


airspace
v=900 (m/s) w=0 (rad/s) 0 Gs BOGIES:1
airspace
v=900 (m/s) w=0 (rad/s) 0 Gs BOGIES:0

(a) static bogies – BASIC MODE (b) moving bogies – ADVANCED MODE

Figure 1 - Simulator, aircraft and path taken over last second (red), bogie and path taken over last second (green) , base station (black), airspace (light blue), test poses (darkblue)

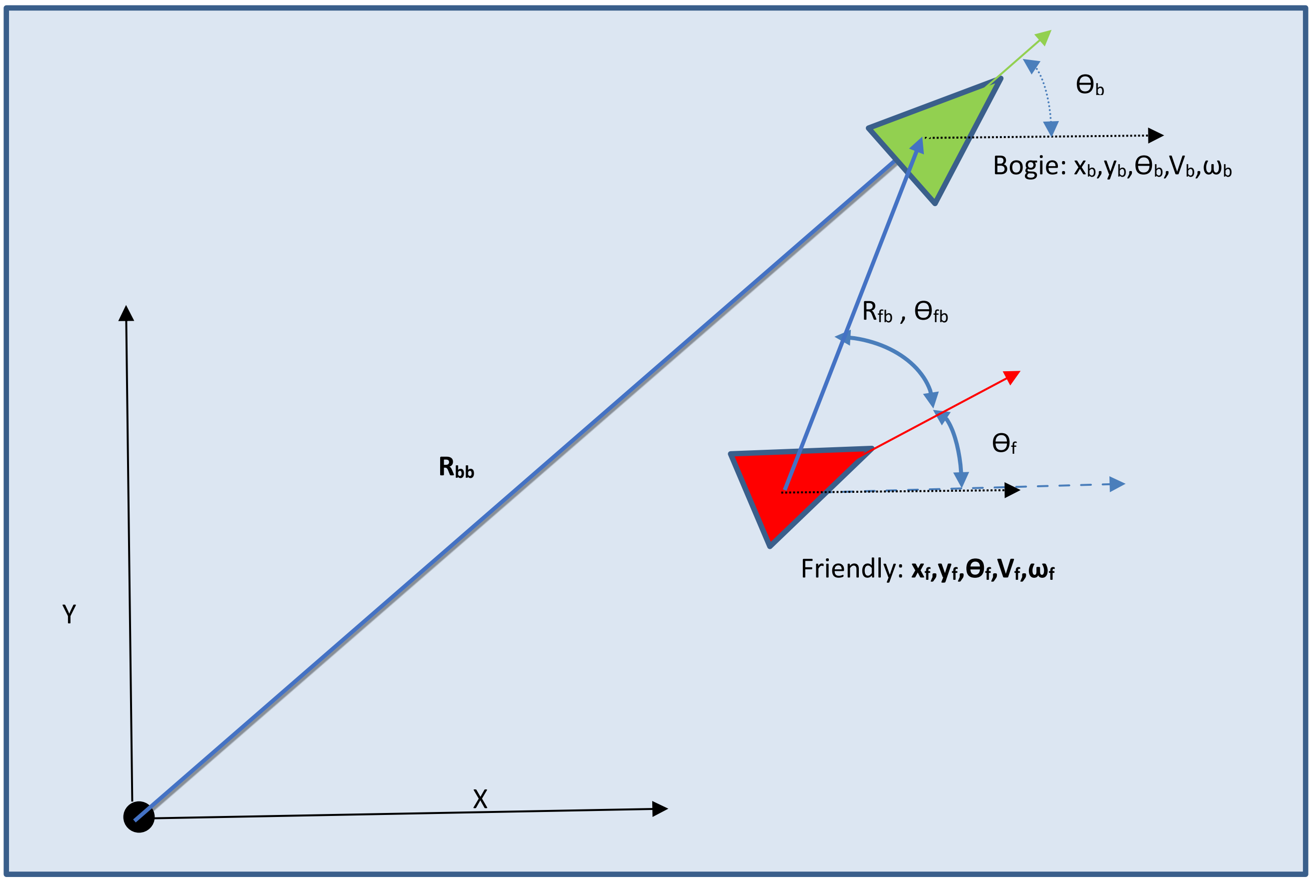

Θ_b
Bogie: $x_b, y_b, \Theta_b, V_b, \omega_b$
R_{fb} , Θ_{fb}
Θ_f
R_{bb}
Friendly: $x_f, y_f, \Theta_f, V_f, \omega_f$
Y
X

Figure 2 – Base station (black), aircraft (red) and bogie (green). The bogie and aircraft can be described with a set of parameters: x,y,ϴ,V,ω [pose: x,y and ϴ][velocity V][angular velocity ω]. The aircfatToBogie readings are related to current position of aircraft (x_f,y_f and Θ_f) – these readings are R_{fb} [range friendly to bogie) and Θ_{fb} (bearing friendly to bogie). The baseToBogie readings are R_{bb} (range base to bogie) and the Velocity of the bogie.

For the aircraft we are provided with the pose, while we can control V and ω.

For the bogie the pose is unknown, the V provided by base station and mostly constant while ω unknown and mostly zero.

The pose of the bogie can be obtained from the data provided, the trajectory of the bogie (successive positions) can be used to estimate V_b and Θ_b

Assessment Criteria Specifics

Criteria	%	**Description / Evidence**
Use of appropriate data structures, locking mechanisms, data sorting mechanisms **Total weight (%) 20** Design adheres to principles of: Inheritance from base class, common data stored and generic functionality implemented solely in base class (no duplication). Member variables/functions that should not be available for instantiation aptly protected. Suitable data containers, sorting mechanisms optimal with respect to memory footprint. Use of synchronisation mechanisms to enable efficient multithreading and safe data sharing between threads (avoiding busy waits / optimise locking).	20	Correct use of access specifiers
	20	Constructors available and used correctly
	10	Containers used to enable time sync of data from multiple threads
	10	Container(s) used to search for an aircraft path [trajectory]
	10	Threading implemented so no loss of sensor data
	10	Use of mutexes makes data secured
	10	Use of convars allows asynchronous operation
	10	Container(s) used to extrapolate future bogie positions
	10	*Base class and derived class are well thought out and justified (BONUS POINTS)*
Proper code execution **Total weight (%) 40** Range data fused (extrapolated/interpolated) and combined with aircraft location to correctly estimate bogie position. Aircraft controlled in specified watchdog timer intervals within limits of aircraft specification envelope (terminal velocity / max G profile). Data fused (extrapolated/interpolated) and combined with aircraft and bogie pose/velocity/angular velocity data to shadow the bogie for time specified. Search for aircraft to be intercepted uses current/future aircraft pose and	15	Aircraft controlled such that (a) watchdog timer does not elapse and (b) commands are not too frequent.
	10	Aircraft controlled so it remains in airspace
	15	Correct position of bogies produced in test poses
	20	Stationary Bogies visited in airspace (at least 12 visited)
	10	Correct pose of bogies produced in test poses
	15	Aircraft controlled with closed loop control. Using point ahead of current bogie pose (has a controller with look ahead point – ie P controller OR pure pursuit) to avoid error

that of bogies in a graph search. [Sections that only apply to ADVANCED MODE HIGHLIGHTED IN GREEN]	15	Search for aircraft to be pursued uses time to impact (closest time to reach target considering relative orientation and velocity of aircraft and bogies)
	10	Number of bogies shadowed in ADVANCED MODE in 5-minute game time is above 40 (BONUS POINTS).
	5	Top three submissions with height number of BOGIES intercepted receive additional BONUS points
Unit tests **Total weight 15%** Unit tests created that guarantee performance of underlying algorithms [Sections that only apply to ADVANCED MODE HIGHLIGHTED IN GREEN]	40	Calculate the bogie position correctly in global space, with reference to base station – the unit tests need to be written in posetest.cpp
	30	Associate the readings from aircraft and base station correctly to bogies (the unit test needs to be written in associationtest.cpp
	30	Calculates the velocity AND position of bogies (extrapolates in timer) for t=3 seconds (the unit test needs to be written in bogieextrapolationtest.cpp
Documentation **Total weight (%) 10**	100	Documentation is produced via Doxygen, all source files contain useful comments to understand methods, members and inner workings (ie extrapolation, control method, search method to target bogie). For HD a landing page (cover page via an index.dox) must be provided with description of submission (what is behaviour of code, what does it use to search/ plan aircraft trajectory)
Modularity of software **Total weight (%) 15** Appropriate use class declarations, definitions, default values and naming convention allowing for reuse. No implicit coupling between classes that disables reuse. All classes interface in ways allowing use of class in others contexts and expansion (ie controlling another aircraft to pursue bogie, adding another	20	Classes designed such that they have sole responsibility to allow reuse
	20	Main ONLY connects objects and commenced threading
	10	Constants are in appropriate locations to allow change
	10	No repeated segments of code that performs same operations
	15	Code can handle varying number of bogies (3 or 7 for example)
	10	Separate threads running control of aircraft and computations of control actions

radar into computation of bogie pose). No “hard coded” values or assumptions. [Sections that only apply to ADVANCED MODE HIGHLIGHTED IN GREEN]	15	The graph in ADVANCED mode can be reused effectively for search looking at minimum time for traversal (reaching all aircraft)

FAQ

What are Radar range limits?	Range of detection 0 to ∞ (infinity)
What is orientation reported in?	Radians, orientation is reported 0 - 2π
Should my assignment have any other classes (header and implementation files)	**ABSOLUTELY**, your main should only setup the threads and setup passing between them. All other implementation should be in classes Consider your classes to have allocated responsibility. Some classes are also needed for computation (especially if it may need to be updated or called reclusively).
How do students generate Doxygen for their assignment	If you wish solely to exploit in text commenting (without adding it to cmake, which suffices for marking) then a quick way is to execute following two commands within your project folder: 1. "doxygen –g" (which will generate Doxyfile) 2. "doxygen Doxyfile"(generates the html and latex) **DO NOT submit the generated document files as part of your ZIP, we can generate them from your source files.**
How to control aircraft / get range	The simulator class provides access to all the data and control utilities of your aircraft, please view example main.cpp. Students are at liberty to use the other classes and structs available for your work (for instance create an aircraft struct and populate readings)
I can't compile the sample code, cmake fails on OpenCVConfig.cmake	If your running A2 on your own device you will need to have Ubuntu and ROS installed. **DO NOT install OpenCV directly from OpenCV website, it is part of ROS!**
Are the number of bogies always limited to 4	Do not make this assumption, your code should work for any number of bogies (3 or 7 for example)