Assignment 3: Threading, Synchronisation and Data Integrity

Revision Date	Change
07-MAY-2020	Initial Release
13-MAY-2020	Added Figure 2 describing the data (parameters to be estimated and those that can be controlled).
19-MAY-2020	Corrected the airspace bounds to be 8kmx8km
20-MAY-2020	Corrected marking criteria

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 in an agnostic manner (b) create a separate thread 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).

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 intercept the aircraft.

The aircraft is controlled by supplying desired linear velocity and angular velocity (V and ω) that the on-board systems immediately responds to, as long as these controls are within operational parameters of aircraft. The controlled linear velocity cannot be less than a terminal velocity (50 m/s), otherwise the aircraft will stall and fall from the sky, or above max velocity (900 m/s). The combination of linear and angular velocity cannot exceed 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 50ms (5ms tolerance), if this is not met the control system will fail (and the application is terminated).

The aircraft is equipment with:

- A high-precision INSGPS that provides the aircraft precise pose (x,y and θ)
- An directional radar that provides range and bearing to the bogies (from the aircraft) updating at 100Hz

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 10Hz (the base station is at location x= 0 y=0).

A library (libsimulator.a) and header files encompassing the simulator is provided (as part of skeleton code – details below).

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/a3_skeleton)**.

The complete API for the Simulator is **available under your git repository (skeleton/a3_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.

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) intercepting these bogies (getting within 200m) of the bogie.

Your system will need to

1. Obtain data from the aircraft and the ground station
2. Utilise the data to estimate the bogie position and display the estimated position on the simulator
3. In order to intercept, linearly extrapolate the bogie position in time (predict where the bogie will be to direct your aircraft - determine what parameters are needed for this task)
4. Estimate a control action that will lead to your aircraft intercepting the bogie
5. Control the aircraft within the watchdog imposed limits and aircraft specification envelope (terminal velocity / max velocity / max G profile).
6. Stay within the workspace around the base station (total area is **8kmx8km** with base station in middle).
7. Intercept as many aircraft as possible

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
2. Creates a suitable number of threads for the task
3. Handle concurrency using appropriate data protection
4. Achieves goals (1-7)

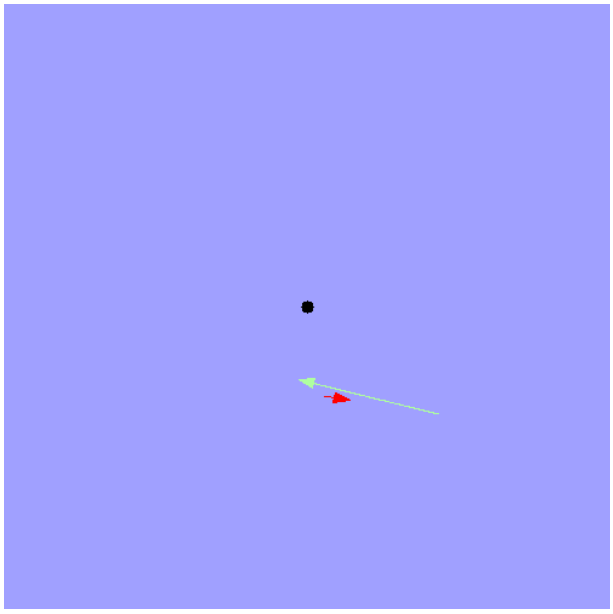

Figure 1 - Simulator, aircraft (red), bogie and path taken by bogie over last second (green) , base station (black), airspace (light blue)

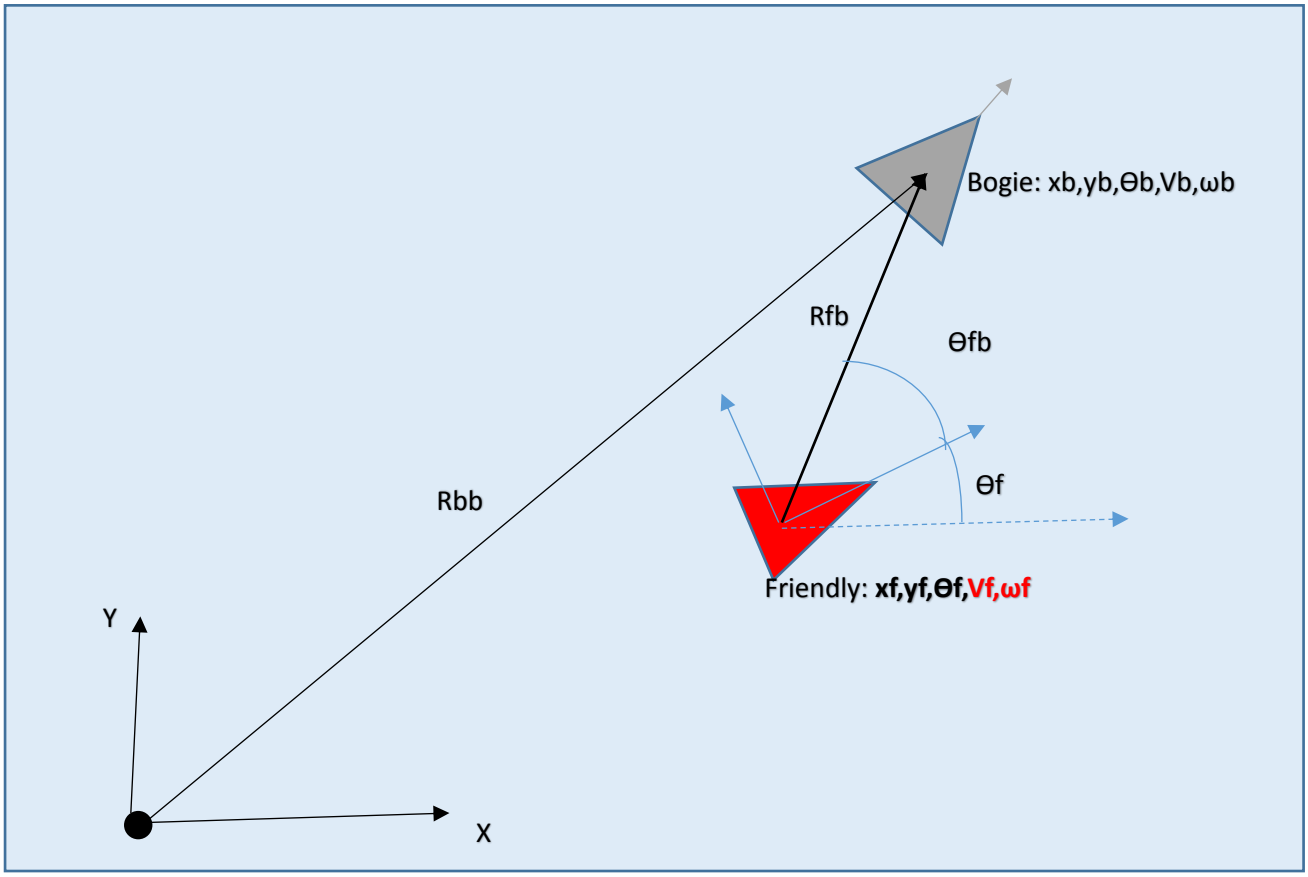

Figure 2 – Base station (black), aircraft (red) and bogie (green). The aircfatToBogie readings are related to current position of aircraft (xf,yf and Θf) – these readings are Rfb [range friendly to bogie) and Θfb (bearing friendly to bogie). The baseToBogie readings are Rbb (range base to bogie) and the Velocity of the bogie. 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]. For the aircraft we are provided 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.

Assessment Criteria Specifics

Criteria	%	Description / Evidence
Use of appropriate data structures, locking mechanisms, data sorting mechanisms **Total weight (%) 40** 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 specifier
	20	Constructors available and used correctly
	10	Containers used to enable time sync of data from multiple threads
	5	Container(s) used to determine velocity of bogie
	10	Container(s) used to extrapolate future bogie positions
	10	Container(s) used to search for a aircraft path [trajectory] (not simply aiming for closest bogie)
	10	Threading implemented so no loss of sensor data AND control of aircraft
	10	Use of mutexes makes data secured
	5	Use of convars allows asynchronous operation
	10	*Base class and derived class are well thought out and justified (BONUS POINTS)*
Proper code execution **Total weight (%) 30** 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.	10	Aircraft controlled such that watchdog timer does not elapse
	15	Aircraft controlled so it remains in airspace
	20	Bogies chased through airspace (aircraft does not simply circle around)
	15	Correct orientation of bogies produced in test poses
	15	Aircraft controlled with point ahead of current bogie pose when chasing (has a controller with look ahead point – ie P controller)
	20	Bogies continuously pursued (aims for new bogie when reaching current one)
	10	Number of bogies shadowed in TOP 10% of

		CLASS (BONUS POINTS)
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). For HD a landing page (cover page) 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 (%) 20** 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 radar into computation of bogie pose). No “hard coded” values or assumptions.	20	Classes designed such that they have sole responsibility to allow reuse
	10	Main ONLY connects objects and commenced threading
	10	Constants are in appropriate locations to allow change
	20	No repeated segments of code that performs same operations
	10	Locking mechanisms embedded with data are activated via function paired with data
	15	Separate threads running control of aircraft and computations of control actions
	15	The computation of path [trajectory] of aircraft could be reused for any other search looking at minimum time for traversal of graph.

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
How do students generate Doxygen for their assignment	A complete introduction to doxygen is provided on UTSOnline as well as a sample project in week 5. If you wish solely to exploit in text commenting (without project files, 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 A3 (skeleton code) on your own device (your not using the VM or FEIT Linux labs) you will need to have Ubuntu 16.04 LTS and ROS Kinetic (or 18.04 LTS and ROS Melodic). You will need ROS for individual projects (A5). **DO NOT install OpenCV directly from OpenCV website, it is part of ROS!** Follow all steps on following link to install ROS (kinetic on 16.04 OR melodic on 18.04) http://wiki.ros.org/kinetic/Installation/Ubuntu http://wiki.ros.org/melodic/Installation/Ubuntu