Assignment 3: Threading, Synchronisation and Data Integrity

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: Sunday 21st May 23:59

Specifics

The physical sensors are collocated; the spatial separation of sensors can be disregarded.

Create a Base Sensor Class (called Ranger) and two derived Sensor Classes (Radar, Sonar). These classes will need to:

1. Store appropriate sensor parameters as per the specifications table
2. Enable querying of hardware specific fixed parameters of the sensor
3. Enable setting configurable parameters of the sensor
4. Inform if the values to be set are sane, use default values if they are not
5. Generate and return range data (r) as per below

   $r = 6 + (4 * \sin\omega t) + \delta$

   where :

   $\omega = 2 * pi * f$ (f=0.01Hz)

   δ is Gaussian noise with mean = 0m and standard deviation = 0.1m.
   http://www.cplusplus.com/reference/random/normal_distribution/]

r is clipped to the maximum specific sensor sensing range

The DataFusion Class will need to

1. On creation be set to a fusion method (min/max/average)
2. Have a method to accept Sensor Data
3. Produce a fusion of the two Sensor Data Streams and
    a. Uses specified fusion method
    b. Deals with readings that are on the boundary of the sensing range (max range).
    c. Linearly extrapolates the sensor data to get the fusion to perform

    For Credit

    d. Produces a fused result at 5 Hz

    For Distinction

    e. Produces a fused result on each new piece of sensor data

Create a Main that

1. Creates an instance of each sensor
2. Allows the user to set configurable parameters of each sensor
3. Starts the sensors generating range data in two separate threads (1 Radar, 1 Sonar)
4. Initialises the Data Fusion class and returns fused data at the required rate until terminated by the user

Sensor 1 – Radar

Specifications

Model	RAD-001
Baud	38400 or 115200
Port	USB (typically /dev/ttyACMX) …where X=0,1,2
Field of View	20 or 40 degrees
Max Distance	10.0m
Min Distance	0.2m
Data Rate	10Hz

Sensor 2 – Sonar

Specifications

Model	SONX-001
Baud	38400 or 115200
Port	USB (typically /dev/ttyACMX) …where X=0,1,2
Field of View	90 degrees
Max Distance	6.0m
Min Distance	0.2m
Data Rate	3Hz

Assessment Criteria Specifics

Criteria	Weight (%)	Description / Evidence
Use of appropriate data structures, locking mechanisms, data sorting mechanisms	40	Inheritance from base class, common data stored and generic functionality implemented solely in base class (no duplication). Classes that should not be available for instantiation are aptly protected. Use of synchronisation objects to enable efficient multithreading and safe data sharing between threads.
Proper code execution	20	Input and querying of sensor attributes works as described. Range data returned is within specification, correctly fused (extrapolated/interpolated), and has the correct timing.
Documentation	20	Documentation is produced via Doxygen. All source files contain useful comments to understand methods, members and inner workings (ie border case handling of fusion, extrapolation).
Modularity of software	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 adding more sensors). No dependency on ordering of sensors, no “hard coded” values or assumptions of sensor order in fusion class.