Assignment 3:

Intent: To analyse a set of specifications, design, test, document and practically evaluate code to perform analysis of sensor data and simple actions based on data on a simulated robotic platform.

Assignment Background

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. Functions that exploit the data require unit testing to ensure they behave correctly. Documentation of your own code allows other developers to utilise it as intended and anticipate outcomes.

Task: Write a series of components using the ROS CBSE framework that will process data originating from a range of sensors on a simulated robot. Employ appropriate multi-threading and data structures to enable time synchronisation and subsequently interrogation of data which allow simple actions of a robotic platform. Supply appropriate auto-generated documentation utilising inline source mark-up. Exploit unit testing framework with test cases evaluating code.

Each project is provided with a standard description (allowing to obtain no higher than Credit level), and a D/HD portion.

BONUS marks are above D/HD, require completion of this component and a theoretical justification (citing a research paper or body of work). The bonus mark CAN NOT be partially awarded.

Support packages and code provided for project

- **pfms_path_following** - this package will drive the robot through a series of supplied pose(s), ensuring each pose is reach (with the orientation supplied), travel between poses is in a straight line.
- Line search between two point in an OgMap is supplied as a library in week 11 (grid processing
- examples of unit tests including rosbags are supplied in quiz 5a / 5b and week 11 material
- a3_skeleton (which is a refactored version of week 11 masterclass, with more comments in CMakeLists.txt and an expanded mainpage.dox)

Project 4: Personal Assistant (Trajectory Following)

The robot assistant (the robot you will control) is robot_0 while the trajectory needed to be followed is robot_1 (we are robot 1 assistant in following him around or standing next to him). The project devises a pose selection strategy, to control the assistant as the follows about, or stands behind / near him in going about the map.

Create a ROS node that:

- Subscribes to the Robot 0 Position, LaserScanner and OgMap on **/robot_0/odom** , **/robot_0/base_scan** and **/local_map/local_map** topics
- Subscribes to the Robot 1 Position **/robot_1/odom**
- Publishes to o **/robot_0/path** topic the pose(s) to go to using the **geometry_msgs/PoseArray** message o **/robot_0/following** topic the poses to go to using the **visualisation_msgs/MarkerArray** message with
    - ✦ ARROW in green as the current goal (pose) location
    - ✦ ARRAOW in blue as any interim poses

The inner working of node is such that it:

For P/C

- Determines when robot_1 is within line of sight and only then does it commence following robot_1
- Continuously determines and publishes poses to drive the robot through while following robot_1 o The pose should lie on the path robot_1 has already taken o The pose should be within line of sight of each other (the line between two poses cannot go over occupied space)
  - A new pose should be sent when you have reached the previous pose
  - If previous poses need to be abandoned send an empty **geometry_msgs/PoseArray** first

Unit tests:

- Determining line of sight between robot_0 and robot_1 using LaserScan and OgMaps (stored inside a rosbag, you can use either data of both combined)
- Sequence of computed poses, to maintain line of sight between poses when following the robot (and he has gone around a corner) on sample OgMaps, LaserScan and odom of robot_0 and robot_1 (stored inside a rosbag)

For D/HD - in addition to above:

Devise an advanced method for following the robot that will always be in line of sight AND 0.5m behind the robot if robot_1 was stationary for a period of time or 0.5m next to him (like an active follower)

- Enable the default or advanced mode by passing a parameter via the node handle
- If robot_1 was stationary for 1-10s compute the pose to be 0.5m behind robot _1 and facing robot_1
- If robot_1 was stationary for more than 10s compute the pose to be 0.5m to the side of robot_1 (on his local frame x-axis : either side) and facing the same direction as robot_1
- Unit test the computation of the poses to go through to be behind OR to either side or robot on sample OgMaps and odom of robot_0 and robot_1 (stored inside a rosbag)

Bonus Mark (additional 15% of marks)

In order to make the waypoint following smoother, directly control the velocity and angular velocity of the robot using steering potential. You can test this style of planning on an area that has only a few objects (cylinder cabinets) and a pose to reach (robot_1 pose).

Assessment Criteria Specifics

Criteria	Weight (%)	Description / Evidence
Use of appropriate data structures, data exchange, data sorting mechanisms, components	20	Access specifiers used appropriately. Inheritance – use of base class as appropriate, common data stored and generic functionality implemented solely in base class (no duplication). Classes that should not be available for instantiation are aptly protected. Constructors for classes set up required member variables. Data stored in STL containers for use, allowing rapid sorting and searching.
Use of appropriate threading, topics and service mechanisms	10	Implements the requested publish / subscribe semantics (topics) and request / reply interactions (services). Responds to incoming data as specified (messages and services contain position in global coordinates). Use of synchronisation objects to enable efficient multithreading and safe data sharing between threads. Effective data management (data does not accumulate endlessly; pertinent data is kept and data not recreated)
Proper code execution	25	Performs functionality with requirements described in P/C and D/HD for each project.
Unit testing framework	20	Unit test of code supplied to validate output for a given input set of parameters.
Supporting Documentation	10	Documentation is produced via Doxygen. All source files contain useful comments to understand methods, members and inner workings (ie algorithm explained). Contains index page that describes how to use the code and expected output.

Modularity of software	15	Appropriate use class declarations, definitions, default values and naming convention allowing for reuse. No coupling between classes that prevents testing and potential reuse) All components interface in ways allowing use of in others contexts and expansion Functionality implemented in Library to Allow Piecewise Testing.
		No implicit “hard coded” values, they need to be configurable from the line when components executed. No assumptions of commencing component (positions of vehicle or map appearance).
Bonus Marks	15	Justification of approach that addresses the challenge Code performs in accordance to presented approach. Unit test supplied.