Assignment 3: Final Project

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.

Individual Task, Weight: 35%

Due: Refer Program in Subject Outline for Due Date

Change Log

15-Oct-2021	Initial Release
25-Oct-2021	Minor changes in document indicated in red, release of a3_skeleton

Table of Contents

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.

Students can select from one of the following projects

1) Project 1: Automated Parking
2) Project 2: Delivery Robot (Travelling Salesman Problem)
3)
4) Project 4: Personal Assistant (Trajectory Following

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)

Oral Defence (VIVA): The assignment has an oral defence (VIVA) that takes place one week after the deadline (Tue-Thu of that week). At the VIVA code design decision are discussed, to determine authenticity and finalise unit testing and execution marks. A booking link will be provided on UTSOnline, the timeslot is 15min per student.

Project 1: Automated Parking

This project determines whether a parking dome, of radius = 1.5m is present within view of onboard laser scanner, and determines interim poses for the robot to move through to get into the dome. The final goal pose is at the centre of the dome (for instance, so the robot is parked inside it and recharging).

Create a ROS node that:

- Subscribes to the Robot Position, LaserScanner and OgMap on **/robot_0/odom** , **/robot_0/base_scan** and **/local_map/local_map** topics
- Publishes to
  - **/robot_0/poses** topic a visualization_msgs/MarkerArray message that shows
    - CYLINDER of 0.2m radius at the dome centre
    - ARROWS as the poses – green are interim poses and red final goal pose.
  - **/robot_0/path** topic the path using the **geometry_msgs/PoseArray** message
- Responds to service requests (of type project_setup/DetectParking) on **/robot_0/detect_parking,** response is
  - (geometry_msgs/PoseArray) computed poses in global coordinates (last pose should be dome centre)
  - std_msgs/Bool indicates if the last pose is dome centre (where the robot shoud park)

For P/C

The inner working of component is such that it:

- Performs dome detection ONLY in response to the service call.
- Transforms coordinates between coordinate systems of local (occupancy grid) and global (robot pose).
- Determines possible domes visible in the laser scanner (the laser returns with intensity=1.0 could belong to the dome, but other cabinets or smaller domes also have high intensity returns).
- Determines interim and final pose (so the entire robot is at the centre of the dome) using the occupancy grid map (so the robot does not hit any objects on it's path to the goal)..

Unit test:

- Detection of dome on sample Laser Scans and OgMaps (stored inside a rosbag)
- Goal pose computation on sample Laser Scans and OgMaps (stored inside a rosbag)

For D/HD - In addition to above

Devise a more advanced method for finding the dome centre location when the opening to the dome is not visible and the robot needs to move to a location to determine the final pose (centre of dome).

- Enable the default or advanced mode by passing a parameter via the node handle
- Publish the pose(s) the robot should move to in order to get a view of the dome in PoseArray in response to service call (with bool indicating last pose is centre = false).
- Unit test your solution
  - Interim pose computation on sample Laser Scans and OgMaps (stored inside a rosbag)

Bonus Mark (additional 15% of marks)

Implement pose selection strategy for the robot to being stored in cabinet (the path should consist of sequence of poses). No additional guidance will be provided for Bonus Mark problem

Project 2: Delivery Robot (Travelling Salesman Problem)

You will be provided a series of positions in global coordinates (nodes) and need to determine a graph that can be built connecting the nodes with edges that ONLY go through free space of the occupancy grid map. Then via a Travelling Salesman Problem (TSP) search, determine the order the nodes will need to be visited such that the total distance travelled to visit all nodes is minimal.

The starting position (start node) is the current position of the robot, the nodes that need to be visited specified are specified via a service call.

Create a ROS node that:

- Subscribes to Robot Position on **/robot_0/odom**, Occupancy Grid on **/local_map/local_map** and **/move_base_simple/goal** (message type geometry_msgs/PoseStamped) topics
- Publishes to
  - **/robot_0/path** topic, the path as **geometry_msgs/PoseArray** message
  - **/robot_0/tsp** topic, the path as **visualization_msgs/MarkerArray** message with
    - CYLINDER Object Type in blue to show the nodes that can be connected
    - CYLINDER Object Type in red to show the nodes that cannot be connected
    - LINE_STRIP Object Type in green as the path
- Responds to service requests on
  - **/robot_0/request_goal** of type project_setup/RequestGoal
    - requests field is the global locations (nodes) that need to be visited
    - response is a sequence of global positions (node) in the order to be visited
  - **/robot_0/cancel** of type std_srvs/Trigger
    - request field is emply and cancels the graph search
    - response field indicates what stage (if any) was completed at time of cancel (further details below)

For P/C

The inner working of node is such that it:

- Transforms coordinates between coordinate systems of local (occupancy grid) and global (robot pose).
- Creates a graph from global position of nodes supplied
  - The graph should be created by joining the nodes that can be connected (the edge between them goes ONLY through free space).
  - The robot position is also a node that needs to be connected
  - If the node cannot be connected it can be removed.
- Searches the graph for shortest travel time between all nodes (after the nodes that are not reachable are removed).
- Abandon current graph construction and TSP if a request via service call **/robot_0/cancel** is made (service type std_srvs/Trigger). The message part of response should be the completed stage of search (NONE, GRAPH or TSP), the success field indicates whether the previous stage was complete.

Unit tests:

- Building Phase of Graph on two sample OgMaps (stored inside a rosbag)
- Query for shortest path on two sample OgMaps (stored inside a rosbag)

For D/HD in addition to above:

While we initially removed nodes that cannot be connected in a straight line, examine if there are nodes that can be connected via a single additional node.

- Enable the default or advanced mode by passing a parameter via the node handle
- Enable connecting nodes of the graph that were initially unable to be connected via a single intermediate node, search the OgMap for the connection points.
- Unit test your solution (show difference in graph resulting from the two solutions)

Bonus Mark (additional 15% of marks)

Reuse your existing graph in the next query by exploiting edges that currently exist between nodes of graph (a node and path that are currently in unknown space (and were once in free space) could be used. This allows to still have paths that were once known to be valid even though the OgMap now does not contain this information.

No additional guidance will be provided for Bonus Mark problem.

Project 3: Path Planning (Rapidly Exploring Random Tree - RRT)

Implement a sampling-based method, Rapidly exploring Random Tree (RRT), to plan paths for the robot to be executed. The approach creates a graph of nodes and edges, expanding the graph with control actions of the robot when it creates edges and additional nodes.

The starting node is the current position of the robot; while the goal position is the desired pose, specified with a Service call or message.

Create a ROS node that:

- Subscribes to Robot Position on **/robot_0/odom**, Occupancy Grid on **/local_map/local_map** and **/move_base_simple/goal** (message type geometry_msgs/PoseStamped) topics
- Publishes to
  - **/robot_0/path** topic the path using the **geometry_msgs/PoseArray** message
  - **/robot_0/rrt** topic a path using the visualisation_msgs/MarkerArray with
    - CYLINDER Object Type in blue to show the nodes
    - CYLINDER Object Type in red for the goal
    - LINE_STRIP Object Type in green as the path (connecting the robot via nodes to the goal)
- Responds to service requests
  - **/robot_0/request_goal** of type project_setup/RequestGoal
    - requests field is the global location (goal) that needs to be visited
    - response is a sequence of global positions (nodes) in the order to be visited
  - **/robot_0/cancel** of type std_srvs/Trigger that cancels the RRT
    - request field is emply and cancels the RRT
    - response field indicates what stage (if any) was completed at time of cancel (further details below)

For P/C

The inner working of node is such that it:

- Determine the configuration space from the OgMap considering the diameter of the Robot (20cm) – that is enlarges the map by diameter of robot so the search does not result in collision positions
- Creates a RRT with basic implementation where the edge is a randomly drawn from actions of going in a straight line either:
  - Forward 1.0m
  - Right 0.4m
  - Left 0.4m
- Terminate the building phase when start and end goal are connected via RRT (where the last node resulting from control action is within 0.2m of goal)
- Abandon current RRT if a request via service call **/robot_0/cancel** is made.
  - Response field indicates how far from the goal the current RRT has reached (string with distance in [mm] ie 2.3m and the bool indicates graph was being grown successfully in last spet (theer were valid control actions)

- Unit tests:
  - Configuration space on two sample OgMaps (stored inside a rosbag)

    - RRT computation on two sample OgMaps (stored inside a rosbag)

- **(For D/HD)**

Rather than expanding the graph with control actions that may not be feasible (the robot does not move sideways left/eight) creates edges that are a result of control actions which are how the robot is controlled, V [m/s] and ω [rad/s].

- use V,ω from the set (0.1,0) (0.25,0) (0.1,0.1) (0.1,-0.1)
- generate control actions with t=2.0s (use delta t=0.1s)
- Enable the default or advanced mode by passing a parameter via the node handle
- Enable changing t for the control actions by passing a parameter via the node handle
- Unit test your solution (show difference in graph resulting from the two solutions)

Bonus Mark (additional 15% of marks)

While our initial RRT goals from starting position to goal, a bi-directional RRT is a significantly faster approach as the graph grows from both the starting and goal position, connecting them both.

NO additional guidance will be provided for Bonus Mark problem.

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
  - **/robot_0/path** topic the pose(s) to go to using the **geometry_msgs/PoseArray** message
  - **/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
  - The pose should lie on the path robot_1 has already taken
  - 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.