CSSR for

Culturally Sensitive Social Robotics for Africa

D4.2.4 Robot Localization

Due date: **26/02/2025**
Submission Date: **15/03/2025**
Revision Date: **n/a**

Start date of project: **01/07/2023** Duration: **36 months**

Lead organisation for this deliverable: **Carnegie Mellon University Africa**

Responsible Person: **Ibrahim Olaide Jimoh** Revision: **1.0**

Project funded by the African Engineering and Technology Network (Afretec) Inclusive Digital Transformation Research Grant Programme		
Dissemination Level		
PU	Public	**PU**
PP	Restricted to other programme participants (including Afretec Administration)	
RE	Restricted to a group specified by the consortium (including Afretec Administration)	
CO	Confidential, only for members of the consortium (including Afretec Administration)	

Executive Summary

Deliverable D4.2.4 Robot Localization focuses on the development of a software module that estimates the pose (position and orientation) of the Pepper robot in a Cartesian world frame of reference. The module achieves this through a combination of relative and absolute position estimation techniques, including odometry, IMU data, and triangulation using visual landmarks. For the relative position, the odometry and IMU data is derived from the Pepper robot's built-in wheel encoders and inertial measurement unit (IMU) sensors, which provide continuous updates on the robot's movement. For absolute position estimation, the module integrates visual detection of ArUco markers as landmarks using an Intel RealSense camera system, enabling correction of accumulated drift and enhancing localization accuracy. The functionality is implemented as a ROS node named `robotLocalization`, which continuously updates the robot's pose in real time. The deliverable outlines the software development process, including requirements definition, module specification, module design, implementation details, running the module, and unit testing. The test evaluates the developed module to ensure its reliability.

Contents

1 Introduction

This deliverable represents the output of Task 4.2.4, aimed at developing a software module for estimating the robot's pose in a Cartesian world frame of reference. The objective is achieved through relative pose estimation methods and absolute pose estimation using visual landmarks. The ROS node `robotLocalization` implements this functionality, providing continuous and real-time pose estimation for the Pepper robot in various operational scenarios.

Accurate robot localization, the process of determining a robot's position and orientation within its environment, is essential for effective and safe interaction with humans and objects, particularly in dynamic social contexts. In culturally sensitive environments, precise localization allows robots to navigate human-populated areas, communicate intentions clearly, perform nuanced interactions without causing discomfort or misunderstanding, and most importantly, arrive precisely at its intended goal [1]. This report addresses the development of a localization module for the Pepper robot in the CSSR4Africa project.

Localization challenges typically include sensor inaccuracies, environmental unpredictability, and cumulative errors or drift from odometry-based methods. To address these issues, the localization module utilizes a combination of relative and absolute localization techniques. Relative localization is achieved through data fusion from the Pepper robot's wheel encoders (odometry) and inertial measurement unit (IMU) sensors, providing a continuous, incremental estimate of the robot's movement. However, these methods alone can lead to significant drift over time, necessitating periodic recalibration.

To mitigate this drift and maintain high localization accuracy, the developed module integrates absolute localization using visual landmarks, specifically ArUco markers. By recognizing these predefined landmarks with an onboard RGB-D camera (RealSense), the robot can periodically correct accumulated errors, ensuring reliable pose estimation over extended periods. The successful integration of relative and absolute localization methods ensures continuous, real-time accuracy of the robot's position.

This deliverable fully documents the localization module development process, providing detailed report into the requirements definition, system specifications, module design, implementation procedures, and testing. Adherence to established project standards specified in Deliverable D3.2 ensures maintainability, reliability, and seamless integration with other system components.

2 Requirements Definition

The robot localization module provide accurate and real-time pose estimation, addressing both position and orientation of the robot in a Cartesian world frame. Relative pose estimation will rely on odometry data and the robot's IMU, while absolute pose estimation will use triangulation based on visual landmarks recognized. The module supports input from RGB and depth cameras, as well as the encoder on the head yaw actuator. Outputs will include the robot's 2D pose (x, y, and rotation about the Z-axis) and annotated images with bounding boxes around detected landmarks.

The primary objective of the robot localization module is to provide accurate and reliable real-time pose estimation to facilitate effective robot navigation, interaction, and task execution within dynamic environments. To achieve this, several key requirements were established.

Firstly, the module must deliver continuous, real-time pose estimation, clearly defining the robot's position as Cartesian coordinates (x, y) and its orientation (yaw angle) within a predefined global coordinate frame. This real-time estimation capability is essential for enabling seamless navigation and precise arrival at its intended goal.

Secondly, the localization module is required to integrate data from Pepper's built-in wheel encoders and inertial measurement unit (IMU) sensors. This approach ensures an ongoing estimation of robot displacement and orientation changes. However, recognizing the inherent limitations of relative localization techniques, especially the accumulation of drift errors over time, the module must incorporate absolute localization capabilities to recalibrate periodically and correct these cumulative errors.

For absolute localization, visual landmark detection through ArUco markers has been mandated due to its effectiveness, simplicity, and robustness. The module must accurately identify these visual markers using the onboard RGB-D RealSense camera, calculate their positions relative to known, predefined locations, and accordingly update the robot's global pose estimate.

Finally, for ease of debugging, validation, and monitoring, the module must provide annotated visualization frames indicating clearly detected ArUco landmarks and the calculated pose estimations. In verbose mode, the pose estimates published to the localization topic and absolute localization update status must be printed to the terminal.

3 Module Specification

The Robot Localization module is specifically designed as a ROS node, `robotLocalization`, that combines sensor inputs to produce precise robot pose estimates in real-time. This section details the input data types, processing methods, output formats, and system configurations involved.

Sensor Inputs

The module accepts three primary inputs: odometry data, IMU sensor readings, and RGB-D images from the onboard RealSense camera. Odometry data provides incremental displacement estimates derived from wheel encoder signals, essential for tracking relative robot movements. IMU sensor data, including linear accelerations and angular velocities, complements odometry by providing a detailed account of the robot's orientation changes.

The third input is visual data captured by the RealSense RGB-D camera system, which is crucial for absolute localization. The module processes these visual inputs to detect predefined visual landmarks (ArUco markers) positioned at known coordinates within the robot's operational environment. By identifying these markers and their corresponding known locations, the module calculates precise corrections to the robot's estimated pose by triangulation.

Output Data

The primary outputs of the localization module include continuous pose estimates, defined by Cartesian coordinates (x, y) and orientation (yaw angle), published on the ROS topic `/robotLocalization/pose`. The module also provides a `/robotLocalization/reset_pose` service to reset the robot's pose using the absolute pose estimation. Additionally, the module provides annotated visual outputs via an OpenCV window, clearly marking detected landmarks within RGB-D camera frames for verification and debugging purposes.

System Configuration

The module reads from a configuration file (`robotLocalizationConfiguration.ini`) that provides platform-specific parameters, such as the sensor and actuator topics, camera type, and pose reset intervals. This ensures the module can be easily adapted to different configurations without altering its core logic.

4 Module Design

The Robot Localization module employs a structured approach that integrates distinct subsystems to ensure accurate and robust pose estimation. The design incorporates two primary subsystems: relative localization and absolute localization, unified through sensor fusion techniques.

4.1 Relative Localization

The relative localization subsystem primarily relies on odometry and inertial measurements. Wheel encoder data provides incremental position updates based on wheel rotations, translating these into displacement estimates. Simultaneously, IMU data, encompassing linear accelerations and angular velocities, helps accurately determine orientation and corrects odometry-derived displacement measurements, especially during rapid movements or when encountering slippery surfaces.

Odometry data processing

The functionality processes odometry data from the robot and applies an initial pose adjustment. This ensures the localization system maintains an accurate and consistent reference frame.

From the received ROS `Odometry` message, the position (x, y) and orientation θ (yaw angle) are extracted as follows:

$$x_{\text{odom}} = \text{msg.pose.pose.position.x} \tag{1}$$

$$y_{\text{odom}} = \text{msg.pose.pose.position.y} \tag{2}$$

The yaw angle θ_{odom} is computed from the quaternion representation:

$$\theta_{\text{odom}} = 2 \cdot \tan^{-1}\left(\frac{\text{msg.pose.pose.orientation.z}}{\text{msg.pose.pose.orientation.w}}\right) \tag{3}$$

To align the odometry readings with the global reference frame, an initial offset is applied:

$$x' = x_{\text{odom}} + \text{adjustment}_x - x_{\text{initial}} \tag{4}$$

$$y' = y_{\text{odom}} + \text{adjustment}_y - y_{\text{initial}} \tag{5}$$

The adjusted position is rotated using a 2D rotation matrix to align with the global coordinate system:

$$x_{\text{current}} = x' \cos(\text{adjustment}_\theta) - y' \sin(\text{adjustment}_\theta) \tag{6}$$

$$y_{\text{current}} = x' \sin(\text{adjustment}_\theta) + y' \cos(\text{adjustment}_\theta) \tag{7}$$

The final global pose correction is then applied:

$$x_{\text{current}} = x_{\text{current}} + x_{\text{initial}} \tag{8}$$

$$y_{\text{current}} = y_{\text{current}} + y_{\text{initial}} \tag{9}$$

The corrected orientation is computed as:

$$\theta_{\text{current}} = \theta_{\text{odom}} + \text{adjustment}_\theta \tag{10}$$

The computed pose (x, y, θ) is published as a `Pose2D` message, with θ converted to degrees:

$$\theta_{\text{deg}} = \theta_{\text{current}} \times \frac{180}{\pi} \tag{11}$$

This ensures the corrected pose is available for localization, mapping, and further processing.

4.2 Absolute Localization

Given the limitations of relative localization methods, particularly the accumulation of drift errors, the design incorporates absolute localization techniques utilizing visual detection of ArUco markers. The absolute localization subsystem is tasked with processing visual data streams from the onboard Intel RealSense RGB-D camera. Computer vision OpenCV techniques identify ArUco markers placed at known locations within the robot's operational space, precisely computing their relative positions to recalibrate and correct the estimated pose provided by the relative localization subsystem.

Landmark-based pose estimation

The functionality processes images from the camera and detects ArUco markers for localization. If sufficient time has passed since the last update, the function extracts marker poses and updates the robot's position by triangulation. Each detected marker provides an estimate of the robot's pose relative to the marker. The detection is performed using:

$$\text{cv::aruco::detectMarkers}(\text{image}, \text{dictionary}, \text{marker\_corners}, \text{marker\_ids}, \text{parameters}). \tag{12}$$

If markers are detected, their poses are estimated using:

$$\text{cv::aruco::estimatePoseSingleMarkers}(\text{marker\_corners}, \text{marker\_size}, \text{camera\_matrix}, \text{dist\_coeffs}, \text{rvecs}, \text{tvecs}). \tag{13}$$

Here, **rvecs** and **tvecs** represent the rotation and translation vectors of the markers with respect to the camera.

Transformation to the global frame

Each detected marker has a known position in the environment reference frame (X_m, Y_m, Θ_m), and its pose relative to the camera is given by (X_c, Y_c, Θ_c). The robot's absolute position is computed by applying a coordinate transformation:

$$X_{\text{robot}} = X_m + \cos(\Theta_m)X_c - \sin(\Theta_m)Y_c \tag{14}$$

$$Y_{\text{robot}} = Y_m + \sin(\Theta_m)X_c + \cos(\Theta_m)Y_c \tag{15}$$

$$\Theta_{\text{robot}} = \Theta_m + \Theta_c \tag{16}$$

The resulting orientation is normalized to lie within $(-180^\circ, 180^\circ)$:

$$\Theta_{\text{robot}} = ((\Theta_{\text{robot}} + 180) \mod 360) - 180. \tag{17}$$

Pose update condition

To avoid unnecessary updates, the robot pose is only updated if the movement exceeds a given threshold (reset interval):

$$d_{\text{moved}} = \sqrt{(X_{\text{final}} - X_{\text{last}})^2 + (Y_{\text{final}} - Y_{\text{last}})^2} \tag{18}$$

$$\theta_{\text{moved}} = |\Theta_{\text{final}} - \Theta_{\text{last}}|. \tag{19}$$

If d_{moved} is below the movement threshold and θ_{moved} is below the rotation threshold, the update is skipped. Otherwise, the updated pose is published as a ROS `PoseStamped` message:

$$\theta_{\text{quat}} = \text{tf::createQuaternionMsgFromYaw}\left(\Theta_{\text{final}} \times \frac{\pi}{180}\right). \tag{20}$$

Algorithm 1 Landmark-Based Absolute Localization (Triangulation)

Initialization
Inputs:
Camera's intrinsic parameters (e.g., K, D) — camera matrix and distortion coefficients.
Known positions of ArUco markers in the global reference frame (LAB frame).
Pose estimates from odometry.
Detection parameters for ArUco markers (e.g., marker size, dictionary).
Outputs:
Estimated robot pose in the global frame: $(X_{\text{robot}}, Y_{\text{robot}}, \Theta_{\text{robot}})$.
Detect ArUco Markers in the Image
Use OpenCV to detect the ArUco markers from the current camera image.
Input: Camera image $I(t)$.
Output: List of detected marker IDs marker_ids and their corner positions marker_corners.
Estimate Marker Pose Relative to Camera
for each detected marker **do**
Use the camera pose estimation algorithm from OpenCV to estimate the pose of the marker relative to the camera:

$$T_m = \begin{bmatrix} R_m & t_m \\ 0 & 1 \end{bmatrix}$$

R_m: Rotation matrix (from marker's orientation to camera)
t_m: Translation vector (position of marker relative to camera)
Use OpenCV's cv::aruco::estimatePoseSingleMarkers() to get R_m and t_m.
end for
Convert Marker Pose to Global Frame
for each detected marker **do**
Use the marker's known position in the global frame and the estimated pose relative to the camera to calculate the robot's global position.
Retrieve the marker position in the global LAB frame: (X_m, Y_m, Θ_m).
Retrieve the camera-relative pose from estimatePoseSingleMarkers(): (X_c, Y_c, Θ_c).
Transform the marker's pose to the global frame using the following equations:

$$X_{\text{robot}} = X_m + \cos(\Theta_m) \cdot X_c - \sin(\Theta_m) \cdot Y_c$$

$$Y_{\text{robot}} = Y_m + \sin(\Theta_m) \cdot X_c + \cos(\Theta_m) \cdot Y_c$$

$$\Theta_{\text{robot}} = \Theta_m + \Theta_c$$

Where:
$X_{\text{robot}}, Y_{\text{robot}}$ are the robot's estimated position in the global frame.
Θ_{robot} is the robot's orientation (in degrees).
end for

Normalize the Robot's Orientation
Ensure the robot's orientation Θ_{robot} is in the range $[-180^\circ, 180^\circ]$:

$$\Theta_{\text{robot}} = \text{mod}(\Theta_{\text{robot}} + 180^\circ, 360^\circ) - 180^\circ$$

Update Pose at Reset Interval
Threshold check: Ensure the robot has moved sufficiently before updating its pose:

$$d_{\text{moved}} = \sqrt{(X_{\text{robot}}^{\text{final}} - X_{\text{last}})^2 + (Y_{\text{robot}}^{\text{final}} - Y_{\text{last}})^2}$$

$$\Delta\Theta = |\Theta_{\text{robot}}^{\text{final}} - \Theta_{\text{last}}|$$

if $d_{\text{moved}} <$ movement_threshold and $\Delta\Theta <$ rotation_threshold **then**
    Skip the update.
end if
Publish the Updated Pose
if the pose has changed significantly **then**
    Update the robot's last pose:
    $X_{\text{last}} = X_{\text{robot}}^{\text{final}}$
    $Y_{\text{last}} = Y_{\text{robot}}^{\text{final}}$
    $\Theta_{\text{last}} = \Theta_{\text{robot}}^{\text{final}}$
    Publish the updated robot pose
end if
Repeat

5 Implementation

The `robotLocalization` ROS node is the core implementation of the module. It processes sensor inputs and outputs the robot's pose in real time. The configuration file specifies parameters such as the platform, camera selection, and reset interval for absolute pose estimation. The implementation adheres to coding standards outlined in Deliverable D3.2, ensuring maintainability and consistency.

5.1 File Organization

The source code for executing the robot localization is structured into three primary components: robotLocalizationImplementation, robotLocalizationApplication, and robotLocalizationInterface. The robotLocalizationImplementation component outlines all the essential functionality required for localizing the robot's pose, both for relative and absolute pose estimation. This component also parse various files necessary for the robot localization functionality such as configuration files and topic files.

The robotLocalizationApplication component invokes those functions from the robotLocalizationImplementation for the execution process of the localization node. The robotLocalizationInterface defines the abstract layer with functions and variables declaration that facilitate communication between the application and implementation layers, thereby ensuring modularity and consistency in the codebase.

The file structure of the robot localization node in the `cssr_system` package is organized as below:

```
cssr_system
└── robotLocalization
    ├── config
    │   └── robotLocalizationConfiguration.ini
    ├── data
    │   └── pepperTopics.dat
    ├── include
    │   └── robotLocalization
    │       └── robotLocalizationInterface.h
    ├── launch
    │   └── robotLocalizationLaunchRobot.launch
    ├── src
    │   ├── robotLocalizationApplication.cpp
    │   └── robotLocalizationImplementation.cpp
    ├── srv
    │   └── SetPose.srv
    ├── README.md
    └── CMakeLists.txt
```

5.2 Configuration File

The operation of the robotLocalization node is determined by the contents of the configuration file that contains a list of key-value pairs as shown in Table 1 below.

The configuration file is named `robotLocalizationConfiguration.ini`

Table 1: Configuration parameters for robot localization node

Key	Values	Effect
`camera`	`FrontCamera, RGBRealSense`	Specifies which RGB camera to use.
`resetInterval`	`<number>`	Specifies the distance that can be travelled in centimetres before the relative pose estimate is reset using the absolute pose estimate.
`robotTopics`	`pepperTopics.dat`	Specifies the filename of the file in which the physical Pepper robot sensor and actuator topic names are stored.
`verboseMode`	`true, false`	Specifies whether diagnostic data is to be printed to the terminal and diagnostic images are to be displayed in OpenCV windows.

Input File

The robot localization node does not read from an input data file.

Output File

The robot localization node does not write to an output data file. The output of the node is returned as publishing the pose estimate to the `/robotLocalization/pose` topic, printing the pose estimate as message on the screen, and landmarks boundary visualization displayed in OpenCV window, if in verbose mode.

Topics File

A list of topics for the robot is stored in the topics file. The topics file are written in the .dat file format. The data file is written as key-value pairs wherein the key specifies the actuators and the value specifies the associated topics. The topics file for the robot is named `robotTopics.dat`

Launch File

The launch file `robotLocalizationLaunchRobot.launch` initializes the robot localization node, the RealSense camera node, and Pepper robot's sensors based on specified configurations. It declares several parameters configured to match your network settings and the robot's configuration, as specified in deliverable D3.3 Software Installation Manual.

Topics Subscribed

This node subscribes to six topics: two RGB camera sensor topics (one published by the Pepper robot, one by the RealSense camera), one depth camera topic (by the RealSense camera), an odometry topic, and inertial measurement unit (IMU) topic, and a joint states topic for the head yaw angle. These are specified in the files identified by the robotTopics key-value pair in the configuration file.

Table 2: Topics the robot localization node subscribes

Topic	Sensor	Platform
/naoqi_driver/camera/front/image_raw	FrontCamera	Physical robot
/camera/color/image_raw	RGBRealSense	Intel RealSense
/camera/depth/image_rect_raw	DepthRealSense	Intel RealSense
/naoqi_driver/odom	Odometry	Physical robot
/naoqi_driver/imu/base	IMU	Physical robot
/joint_states	Head Yaw	Physical robot

Topics Published

The robotLocalization node publishes to topics as listed in Table 3 below.

Table 3: Topics published by the robot localization node.

Topic	Node	Platform
/robotLocalization/pose	gestureExecution overtAttention robotNavigation behaviorController	Physical robot

Services Supported

This node provides and advertizes a server for a service /robotLocalization/reset_pose to reset the pose of the robot using absolute pose estimation. It uses a generic msg, Reset.msg with just one field string, with a value "reset". If the reset request is successful, the service response is "1"; if it is unsuccessful, it is "0".

Table 4: Services supported

Service	Message Value	Effect
/robotLocalization/reset_pose	reset	Reset robot pose.

6 Running the Robot Localization Node

Running the `robotLocalization` node requires modules including a set of clearly defined ROS launch files and configuration parameters. To initiate the robot localization module on the Pepper robot hardware, you must ensure that all necessary sensor drivers and ROS components are correctly installed as outlined in deliverable D3.3 Software Installation Manual.

NOTE

Ensure that the RealSense camera is properly connected to the Pepper robot and operational. To visualize its feed, you can run:

```
rosrun image_view image_view image:=/camera/color/image_raw
```

To visualize the RealSense camera calibration using ROS (intrinsic parameters), run:

```
sudo apt-get install ros-<your_ros_distro>-camera-calibration
```

replace $< your\_ros\_distro >$ with your ROS distribution, and:

```
rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.024
    image:=/camera/color/image_raw camera:=/camera/color
```

Launching the localization node

Launching the node is achieved using the launch file `robotLocalizationLaunchRobot.launch`. This launch file encapsulates all parameters, node configurations, and sensor initializations required for the localization system's full functionality. To execute the launch file, users should run the following command in a terminal:

```
roslaunch robotLocalization robotLocalizationLaunchRobot.launch
```

The launch file initializes the localization node alongside ROS subscribers and publishers necessary for sensor data acquisition and localization outputs.

Verifying successful startup and running

To verify successful startup and running of the node, users should monitor the terminal outputs, looking specifically for initialization confirmation messages from the ROS topic `/robotLocalization/pose` and visual data topic from the RealSense camera system node in an OpenCV window.

If the node runs successfully, the message `"robot localization node running"` is printed to the terminal at every interval. When the absolute pose estimate is updated, the following message `"absolute pose updated successfully:<x_position> <y_position> <theta>"` is printed to the terminal.

To view the pose estimate of the robot in a continuous stream, visualize the topic by running:

```
rostopic echo /robotLocalization/pose
```

7 Unit Testing

The unit testing is designed to validate all individual components and integrated subsystems of the robot localization process, and adhering to the standards outlined in Deliverable D3.2 Software Engineering Standards Manual and Deliverable D3.5 System Integration and Quality Assurance.

The file structure of the robot localization unit test is as follows.

```
unit_test
└── robotLocalizationTest
    ├── config
    │   └── robotLocalizationTestConfiguration.ini
    ├── data
    │   └── testTopics.dat
    ├── include
    │   └── robotLocalizationTest
    │       └── robotLocalizationTestInterface.h
    ├── launch
    │   └── robotLocalizationTest.launch
    ├── src
    │   ├── robotLocalizationTestApplication.cpp
    │   └── robotLocalizationTestImplementation.cpp
    ├── srv
    │   └── SetPose.srv
    ├── README.md
    └── CMakeLists.txt
```

To verify the correct acquisition and preprocessing of sensor data inputs, isolated tests were conducted. Individual ROS subscribers for odometry (`/pepper_robot/odom`) and camera inputs (`/camera/color/image_raw` and `/camera/depth/image_rect_raw`) were tested.

Test scenarios included simulated sensor data streams with known properties to ensure each data channel's correct parsing, synchronization, and preprocessing. These tests ensured that each subscriber reliably acquired data at expected rates and correctly handled sensor-specific anomalies such as data dropouts or inconsistent message formats.

Tests were also performed on the visual landmark detection subsystem. Introducing several ArUco markers placed at known positions in the environment and the camera's ability to detect any in its range of view to compute the robot's pose.

Table 5: Test cases

Test Case	**Description**
odometry	Verifies that the odometry data is published by the Pepper robot and the subscriber node is able to subscribe to the topic.
cameraInputs	Verifies that the RealSense camera system publishes the camera topics and the camera feeds can be visualized in an OpenCV window.
detectArucoMarkers	Verifies that the ArUco markers in the environment can be detected and the pose can be estimated based on the ArUco markers in the camera's field of view.
updateAbsolutePose	Verifies that the robot's pose is correctly updated at specified intervals on the `/robotLocalization/pose` topic.

The results of the unit tests falls under success or failure conditions. If a test is successful, a success message is logged, otherwise, a failure message is logged.

References

[1] Gabriele Trovato, Massimiliano Zecca, Salvatore Sessa, Lorenzo Jamone, Jaap Ham, Kenji Hashimoto, and Atsuo Takanishi. Cross-cultural study on human-robot greeting interaction: acceptance and discomfort by egyptians and japanese. *Paladyn, Journal of Behavioral Robotics*, 4, 12 2013.

Principal Contributors

The main authors of this deliverable are as follows:
Ibrahim Olaide Jimoh, Carnegie Mellon University Africa.
David Vernon, Carnegie Mellon University Africa.

Document History

Version 1.0

First draft.
Ibrahim Olaide Jimoh
15 March 2025.