Autonomous Robotics (AURO) Report

I. DESIGN

Leveraging the robustness of ROS2 and the precision of the NAV2 navigation stack, this system will stand as a technical embodiment of the project's goals as outlined in the assessment task.

System Architecture

Through the employment of paradigms such as modular and layered architectures, our autonomous robotic system adopts a hybrid architecture of the layered modular system-architecture, combining the best attributes of both paradigms. The architecture provides a structured yet flexible approach through the division of the system into distinct layers, each consisting of interchangeable and independent modules dedicated to perform specific tasks. The modularity of these layers promotes reusability and ease of maintenance, because it cuts down on computational complexity with the abstracting of repeatable processes and segregates processes for inspection and implementation. Along with this, the layered structure ensures an organised hierarchy of data flow and command execution.

To understand this architectural design more clearly, we must first design the layers and their modules:

- Control Layer: Topmost layer, dedicated to high-level decision making; encapsulating the systems strategic logic and operational state machine.
- Perception Layer: Middle layer, responsible for interpreting sensory data and assessing the robot's environment.
- Navigation Layer: Foundational layer, for dealing with robot movements and interaction with the physical world.

Component Roles

The *Robot Controller* (Control Layer) is considered the brain of our system, processing input from the perception layer to make decisions about navigation and task execution. Orchestrating the flow of operations using a finite state machine, determining if the robot should be collecting an item, returning home or engaging in exploration behaviours for better vantage of items. With the NAV2 stack, navigation is simplified, distributing heavy computation to dedicated services such as localization, path planning and collision avoidance. This will allow for faster computation within the control layer, making the robot more efficient and more responsive to new information.

The *Item Assessment* (Perception Layer) is a key module, which utilizes vision processing, from hardware such as Camera Image Recognition and Lidar Sensors, to evaluate and prioritize items in the environment. With the robot's positional data, it will determine the cost of retrieving an item based on either the item's intrinsic value or the distance away from the robot in comparison to another item or home. Using the concept of geometric optics and trigonometry, the module leverages the camera's field of view (FOV) and focal length, along with the known diameter of items (0.15) to precisely calculate the distance and angle to them. Achieved through the comparison of the image size of the item and its actual size, to give an accurate estimation of each item's location relative to the robot. Ensuring that the decision-making process for item retrieval is both efficient and informed.

As part of the navigation layer, the *Random Goal* module is invoked when the robot needs to reposition strategically. Using spatial algorithms to determine the direction best suited for the robot to navigate resulting in a wider range of possible positions (avoiding map boundaries), a random position is generated defined by the robot's current position. Typically, this module should be invoked when no viable options are available for item retrieval.

Acting as a crucial component within the navigation layer, the *Robot Position* node will continuously update the entire system with the robot's precise position based on transformations from the robot's *base_footprint* frame to the *map* frame (world origin). This will be explored further in the implementation, as it relies upon fundamental characteristics of the ROS2 framework.

Independent of the layered architecture is the *Data Logger*, an integral to the system for analysis, which records the total items collected and overall score, given as a table in a csv file. This was supplied by the assessment, requiring little modification except for post-performance analysis.

Interactions

The interactions between each component are as follows:

- The *Robot Controller* acts on information received from the *Item Assessment*, dictating whether to pursue an item and its cost. As well as information from *Random Goal* when triggered for repositioning the robot for better vantage of items.
- The *Robot Position* feeds real-time positioning data to both the *Robot Controller* and *Item Assessment* for navigational and item assessing decision-making. Where it is passed through to *Random Goal* within the *Robot Controller* when repositioning is required.
- The *Random Goal* influences the *Robot Controller* as stated prior by suggesting new navigation points, to be executed through the NAV2 stack's path planning capabilities.

Design Rationale

The decision to opt for a layered modular architecture is driven by the system's need for robustness and adaptability. Each layer is designed to function independently, ensuring any changes or failures in one module have minimal impact on the rest of the system. This separation not only simplifies the development and maintenance, but also enhances the system's ability to evolve and integrate with new technologies or algorithms over time.

ROS2 as the middleware framework for this project, is not only a requirement of the assessment, but also provides excellent flexibility and a set of powerful tools tailored to robotics applications. This is shown through the decision to

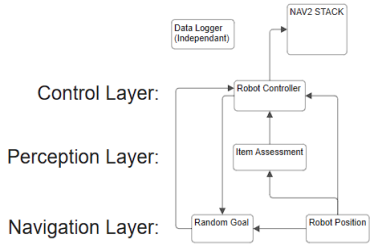
use the NAV2 stack, as it offers sophisticated navigation capabilities out-of-the-box, allowing for the *Robot Controller* to focus on higher-order decision-making rather than an abundance of high computation complexity operations.

Alongside this, ROS2 uses topics, services and actions to communicate between nodes; paired with the modularity of the system, this ensures a smooth data flow and efficient command execution dealing with new data as it's published and subscribed to. These choices of communication make to balance the need for real-time responsiveness and overall operational complexity.

The direction to use geometric optics and trigonometry for item analysis (distance and angle calculations), ensures precise and reliable information for item retrieval and navigation. Meaning there is no need to continuously update the item position for the robot to navigate to, as it will be accurate every time with minimal margins for error.

The design of the *Random Goal* is also particularly notable, using calculating movement goals based on open-space detection, because it ensures the robot does not simply wonder aimlessly but instead moves with the purpose of moving to a more optimal search environment.

In summary, the design of our robotic system is the result of careful consideration of the operational environment, tasks to be performed and available technologies with ROS2. And with the combination of the systems architecture and ROS2 with NAV2 positions the system for the ability to be effective with the dynamic approach to item retrieval and navigation, as well as potential for future growth. See below for a high-level diagram of the proposed system design:



II. IMPLEMENTATION

At the heart of the implementation is the *Robot Controller*, which utilizes the foundational element of ROS2: Node Structure. This facilitates communication with other nodes in the network, using topics and services. Through these topics and services, the *Robot Controller* node can stay well informed for making decisions and trigger vital computation when required: such as *Item Assessment* and *Random Goal*. Initialized with the key parameters from the launch script, robot name and initial pose; fundamental to operations such as checking if the robot is carrying an item and it's starting position for navigation using the NAV2 stack navigator. This produces, manages and sends the path planning to the robot for reaching a given goal. The node subscribes to topics '/item_holders' and 'items' which are supplied by nodes within the assessment publishing information about robot's that are holding items, as well as a list of items that can be seen within the camera's field of view. A third subscription to '/robot_position' topic reads real-time information about the robot's position using transforms. This data is crucial to the finite state machine (FSM) used by the *Robot Controller* to decide the correct state to operate. The FSM is a dynamic framework that guides the robot through various operational states: CHECK_ITEMS, FIND_NEAREST_ITEM, SET_HOME, SET_SEARCH, NAVIGATING. Which is implemented within a control loop and timer to iterate through the FSM every 1ms at a rate of 100Hz.

In the CHECK_ITEMS state, the *Robot Controller* determines if the robot is currently holding an item, through the subscription to the '/item_holders' topic which checks against the data to see if there is a match to the robot's name provided in initialization, done using a callback procedure. If the robot is holding an item, then the robot should move to go home, transitioning to the SET_HOME state in the subsequent iteration of our control loop. If the robot is not holding an item, this state will use the subscription to data in the '/items' topic to determine if there are detected items on the robot camera. If there are, the robot sends a request to the *Item Assessment* service with information regarding the list of available items and previous item goal value (typically 0, unless a previous item retrieval failed). Before, transitioning to the FIND_NEAREST_ITEM state in the following iteration. Finally, if the robot is not holding an item and no viable items can be seen, the robot sends a request to the *Random Goal* service with the robot's current position to obtain a randomized goal vector; before transitioning to the SET_SEARCH state in the next iteration.

Within the FIND_NEAREST_ITEM state, once the request has been processed by the *Item Assessment* service, the response is extracted to reveal the nearest/best possible item, it's distance from the robot and value. Using this information, we can calculate the exact position of the item in the environment. This is done using camera optics, where we know based on the model files that the robot's camera has a FOV of 1.085595 radians, and an image width of 640pixels. The focal length can then be computed using $f = \frac{w_{image}}{2*\tan(\frac{FOV}{2})}$, where w_{image} is the width of the image as found in [1]. Then calculate the angle of the item in relation to the robot using the x position of the item on the screen against f, before adding the robot's current rotation (yaw). Then using the equation $\theta = (rel_{angle} + \pi) mod_{2\pi}$ to normalize the angle between $-\pi$ to π.

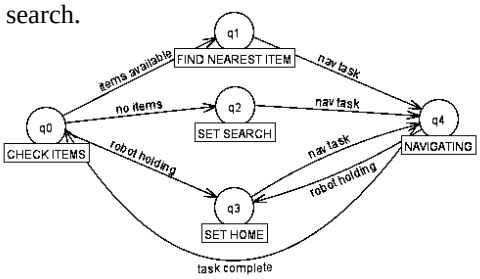
Using this information, we can determine the exact position on the map where our item is located, to create a PoseStamped to be passed to the NAV2 stack navigator that will handle path planning and collision avoidance towards the item. This is calculated using Cartesian coordinate conversion: $x_2 = x_1 + r * \cos(\theta)$ & $y_2 = y_1 + r * \sin(\theta)$. Once the goal position has been given to the NAV2 navigator, the NAV2 stack will begin creating a task to navigate to the given goal based on the given map file for localization. Following this, the FSM will transition to the NAVIGATING state in the next iteration of the control loop.

The SET_HOME state is only transitioned to when the robot is holding an item or if the amount of time the simulation has been running exceeds a given time (300secs). Within this state we create a point in the known boundaries of the home area according to the map file: x_{min}, x_{max} (-4.0, -3.0) & y_{min}, y_{max} (-3.0, 3.0) to create the nearest point relative to

the robots position in the home zone. The approach taken was that a direct route using the current y position and the x position to be set as -3.5: a central line across the home zone to ensure the robot is within the bounds to count as item retrieved. Once the home goal has been set and passed to the NAV2 navigator to start navigation, the FSM is transitioned to the NAVIGATING.

The SET_SEARCH state is transitioned to when no options are available for item retrieval, remaining in this state until a response has been received from the *Random Goal* service. This response contains the x, y and θ to be used to set the new goal for the NAV2 navigator to use for path planning, and the orientation. Once set, the FSM will transition to the NAVIGATING state.

The NAVIGATING state of the FSM, where the focus is on managing and monitoring active navigation tasks. The robot first checks if a task is still running; retrieving feedback from the navigator, including the estimated time of arrival and logging it for reference. Should the navigation exceed a pre-determined duration (30secs) before the task is canceled. Upon completion either by success, cancellation or failure, the state will process the result and determining the next step based on its result. If succeeded, a check for a holding item is performed, resulting in a transition to SET_HOME if true, and CHECK_ITEMS otherwise. In the case of cancellation and failure, a log is taken, and the state is transitioned to CHECK_ITEMS to prepare for either a new item retrieval or random search.



As mentioned, the *Item Assessment* and *Random Goal* are services. Where services "are based on a call-and-response model … only providing data when they are specifically called by a client" [2], this creates a more efficient system, because it reduces the amount of continuous computation needed to function. Instead, only computing when necessary.

Looking at the *Item Assessment*, this service node is initialized with parameters for focal length (f) and actual diameter of items, which are essential for calculating the distance to each item using computer vision. The formula below is derived from the principles of optics and similar triangles, as discussed in [1]. Where $\frac{f*actual\_diameter}{image\_diameter}$ can be used to calculate the distance to an item, assuming a linear relationship between focal length and diameter. Upon receiving a request, the service iterates through each item in the provided list, calculating distance and comparing its value to a threshold. Including comparisons to best item value, to select an item that exceeds the value threshold and is closer than any previous considered item. And in response to this request, the service will send the best item along with the distance to the chosen item.

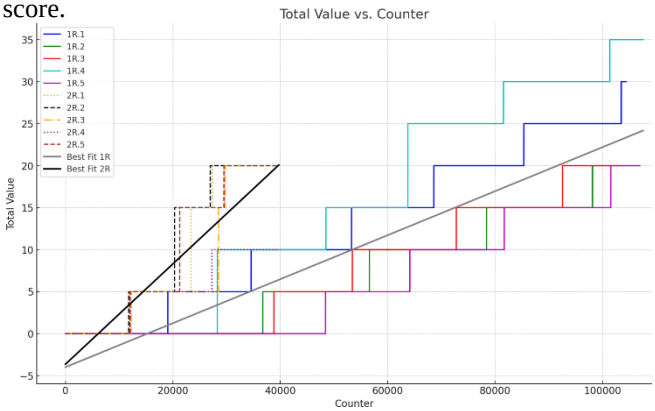
The *Random Goal* service calculates a goal position by dynamically computing a random angle and distance based on the robot's position and orientation. Within the service request, the service receives the positional data and determines the random angle using a series of conditional statements. The statements determine the most appropriate direction for the robot to travel for more possible movements. For instance, if the robot is positioned in the upper-right corner, the service will determine the best angles for traversal are between east and south. Ensuring the robot remains in the boundaries; then using Cartesian coordinate conversion, calculates a random position at a random distance (1-1.5units) as a response to the client. This dynamic goal setting is crucial for developing sophisticated robotic systems capable of operating in complex environments.

Unlike in our initial design, it is apparent that the robot position is not directly subscribed to from each node. Instead, the *Robot Controller* subscribes to the topic '/robot_position' which receives data from the *Robot Position* node, to then use in its respected interactions with service providers. To do this, we use TF2 (a transform tool within ROS2) [3], which can calculate the transformation between frames and their positional vectors (ex. *odom*, *map*, *base_footprint*). For instance, to calculate an accurate robot position a transformation between the *map* frame (origin) and *base_footprint* frame is required. This transformation gives the (x,y) position of the robot, and using another ROS package 'tf_transformations' [4] we can calculate the (roll, pitch, yaw) using the *euler_from_quaternion* function.

III. ANALYSIS

The experimental approach was to use the data received from the independent component: *Data Logger* supplied by the assessment to record the types of items collected, their score and the overall score of the simulation over time using the counter (Sim Time). Crucial to post-performance analysis, where we can identify trends and calculate efficiency of the system based on a quantity of test runs. For this post-performance analysis, it was opted to run the system using a virtual machine with all necessary tools installed onto Ubuntu 22.04. With the system specification: 8GB RAM and 4 dedicated CPU Cores from the host machine. Note, that due to the limited resources available it was only possible to test a single robot simulation and a multi-robot simulation of 2robots. Otherwise with the dedicated 5minutes (real-time) to each run, the 3robot system would not have adequate time to establish results.

The graph below, indicates the resulting score over time for 10 runs: 5 runs for each sim variation. The x-axis denotes the time as a counter, and y-axis as a measure of cumulative score.



The results compare the performance of single (1R) and multi-robot (2R) simulations with respect to simulation time (Counter), to reveal distinct behaviours between the two setups within a fixed 5-minute real-time constraint.

In the single robot simulations, the total value increases in a stepwise manner, due to periods of navigation to an item and returning. The overall trend, as depicted by the "Best Fit 1R" line, indicates a consistent but relatively gentle increase in score over time; a steady, albeit slower rate of value accumulation than "Best Fit 2R". The extended counter time for these simulations imply that they were less taxing on the systems resources, enabling longer simulated time within the real-time limit. Comparatively, the multi-robot simulations (2R) demonstrate a more pronounced increase in total value, with the "Best Fit 2R" line exhibiting a steeper increase over time. This implies that the simultaneous operation of multiple robots leads to a more rapid accumulation of value. However, the shorter counter times for 2R scenarios prove the need for higher resources due to the increased computational load of managing multiple robots. Significantly constraining the simulated time achievable within the real-time period.

In summary, while multi-robot simulations have the potential for higher efficiency in terms of value gained over simulation time, they require more computational resources. However, this analysis of data proves through the positive gradient, that this system is efficient and effective at item retrieval over time given the proper resources.

IV. EVALUATION

As the autonomous robotic system is powered by ROS2 and NAV2's navigation stack. Equipped with a structured and modular architectural approach, the system demonstrates a strong design for executing tasks in an organised manner. Paired with the positive trends observed in the simulation data analysis, it confirms the system's skill in item retrieval tasks and validating the dynamic control facilitated by the FSM implementation.

As stated, a key strength lies in the systems architectural design, promoting the ease for maintenance and systematic debugging: essential for real-world applications. The modularity in the systems design allows for clear separation of functionalities; enhancing the system's ability to adapt to new tasks and technologies. This adaptability is crucial, as displayed in the simulation results; proving the systems capability to navigate and complete tasks in a controlled environment. For instance, the use of principles in camera optics and computer vision shows that the system is already reaching for more advanced technologies to increase the efficiency of task execution.

However, despite these strengths, the system exhibits limitations in scalability and resource efficiency: particularly in multi-robot simulations where the simulated time is constrained by computational demand. Indicating potential challenges in deploying multiple robots simultaneously in real-life scenarios where resources are finite and costly. Furthermore, the execution used for item identification and localization is simplistic and does not account for lens distortion or other worldly factors that would affect the accuracy of the robot's calculations. For example, if an item moved midway through navigation to the goal, the robot would not know until it reaches the original position to see it has been moved. The transition from simulation to real-world would necessitate rigorous testing in diverse conditions to bolster the system's reliability. Enhancements in error handling, and development of adaptive algorithms would be vital to mitigate the effects of sensor inaccuracies and environmental unpredictability. Fortunately, the system's flexibility and modularity provide a solid foundation to develop these enhancements from.

In summary, the system's design and implementation demonstrate substantial promise for precision and adaptability in controlled environments. Yet, for real-world deployment, it requires advancements in handling real-world complexities and computational efficiency. With focused improvements on these fronts, the system is well-positioned to be an effective solution for real-world autonomous tasks.

V. SAFETY AND ETHICS

When devising an autonomous robotic system, safety and ethical concerns are crucial. Safety-wise, our system includes collision avoidance capabilities, however due to real-world unpredictability it necessitates advanced safety protocols and adherence to regulatory standards for protecting humans and property. Ethically, the introduction of autonomous systems must consider potential job displacement, ensuring transition for affected workers or provided training towards newer positions. As well as, considering the upkeep of privacy with data-sensitive components such as cameras. Furthermore, the autonomous decision-making processes embedded into the system must be transparent, avoiding bias and ensuring equitable actions, especially in critical resource allocation.

Through our system's modular design, there is the immediate opportunity to incorporate safety mechanisms directly into operational layers. For instance, the *Robot Controller* can be programmed with emergency stop functions and prioritization of human safety in its decision-making, such as constant checks within the navigation stage for unpredictable changes in the environment. Ethically, the *Data Logger* can be updated to adhere to data protection and handle the data responsibly, respecting privacy and transparency.

Real-world implementation would involve constant and iterative refinements to maintain ethical integrity and safety, underpinned by stakeholder engagement to align the system's operation with societal values and norms. Ensuring the systems efficiency is matched by its responsibility to ethical and safety standards.

REFERENCES

[1] R. Szeliski, "Computer Vision: Algorithms and Applications," 2nd ed., 2022. https://szeliski.org/Book/.
[2] "Understanding ROS2 Services" ROS2 Documentation - Humble. https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html.
[3] "TF2 Tutorials" ROS2 Documentation - Humble. https://docs.ros.org/en/humble/Tutorials/Intermediate/Tf2/Tf2-Main.html.
[4] D. V. Lu, "tf_transformations package," GitHub. https://github.com/DLu/tf_transformations.