Autonomous housekeeping

Johan Nordling
Linköping University
johno762@student.liu.se

Lukas Olsson
Linköping University
lukol280@student.liu.se

Jonathan Öhrling
Linköping University
jonoh749@student.liu.se

Robin Simonsson
Linköping University
robsi315@student.liu.se

Max Rüdiger
Linköping University
maxru105@student.liu.se

Abedalhkeem Najeeb
Linköping University
abena406@student.liu.se

Abstract

In this project, an autonomous robot was developed using Turtlebot 4 to build and maintain a useful worldview of a laboratory environment. The goal of the project was to demonstrate advanced AI and robotics capabilities and their potential applications to Trafikverket, the Swedish transportation agency. To achieve this, the robot was to be equipped with the ability to create a 3D model of the environment, detect and classify objects in the environment, track the location of objects, and patrol the environment. ORB-SLAM2 and Nav2 were used to map and navigate the environment, while YOLOv7 was used for real-time object detection. Object tracking was to be implemented by maintaining a database of detected objects and their locations, and updating the database as the robot navigated the environment. The project assigner did not prepare properly, so while a method for the project was developed, it was not implemented on the Turtlebot. Instead, only a smaller proof of concept code base was created to run on pre-recorded ros2bags.

1. Introduction

This project aimed to realize an autonomous robot with the capability to build and maintain a useful worldview of the lab premises using a Turtlebot 4. The finished project was expected to be shown to Trafikverket higher-ups to show them what advanced AI and robotics can do and how it can be applicable to their use cases. The expected project goals were the following:

- Build (and maintain) a good 3D model of the lab, that can be used in for example VR/AR applications.
- Discover if an object has been moved, is missing or has been added to the environment.
- Be able to find objects as requested by a user, i.e. by telling where it is or by moving there and showing.

These goals were broken down into a few practical problems that had to be solved:

- Map the environment in a suitable way for navigation and localization
- Navigate the environment
- Detect/classify objects and keep track of them in a 3D space
- Patrol the environment and scan for object
- Generate a textured mesh of the 3D environment

1.1. Possible Solution

1.1.1 Mapping and Navigating the Environment

The first problem that had to be solved was the process of mapping the environment for navigation and localization. After conducting research, several solutions to this were found. And it was concluded that ORB-SLAM2[2] together with Nav2[6] would be the best solution to this problem.

ORB-SLAM2 is a library used to capture camera trajectories and create a sparse 3D reconstructions of an environment. It manages loops and relocalization of the camera in real-time and has tight integration with ROS2 and Nav2. This is what made us choose ORB-SLAM2 for this project.

Nav2 is a navigation library for ROS2 that can be used in conjunction with ORB-SLAM2 to navigate while capturing the environment. This provides us with a well-integrated system for navigating the environment and thus leading to the decision to use this library for the project.

ORB-SLAM2 together with Nav2 served as the foundation for this project and was relied upon by the rest of the system.

1.1.2 Real-time Object Detection

The second main problem that had to be solved was detecting 3D objects in the environment. There are several ways to approach this problem but since this project is limited in scope, the chosen solution of using a 2D classifier together with the 3D data given by our navigation and localization system to mark objects in 3D.

The chosen 2D classifier was YOLOv7 [12]. Our reasoning behind this was based on the accuracy and the speed of the classifier as well as the existence of pre-trained models. A fast solution with decent accuracy was desired due to the need for real-time processing, with a focus on the location of objects rather than nuanced classification.

1.1.3 Dynamic Object Tracking

Object tracking, in the sense of knowing where objects are, was accomplished by keeping track of what objects have been detected and what objects were expected to be seen. The detected objects are stored in a database along with a timestamp and their 3D position. If a similar result already exists in the database, based on proximity to the estimated position, it is assumed to be the same object and is ignored. As the environment is navigated, it is checked whether the object is still present. If it is no longer visible, it is removed from the database, but if it is still visible, the timestamp is updated.

1.1.4 Textured Environment Mesh

There are several ways to generate a textured 3D mesh of an environment depending on what the model should be used for. If the model is intended to be used in VR/AR applications a good solution to capture an environment would be a library such as Voxblox.

Another option would be to generate a 3D model based on the data provided by ORB-SLAM2. However, this seems to yield results that are worse than Voxblox, which makes sense since ORB-SLAM2 is meant to capture/calculate the camera trajectory and a sparse 3D reconstruction of the environment rather than an accurate 3D model reconstruction.

2. Method

2.1. System and Environment

The hardware used requires the use of a virtual environment and Docker was chosen for this task. The Docker environment includes dependencies for development on the TurtleBot 4 platform such as the Robot operating system (ROS) and tools to help in development. To set up this environment correctly the tutorial on the Turtlebot website should be followed. This will also ensure that all the dependencies needed will be included in the Docker environment.

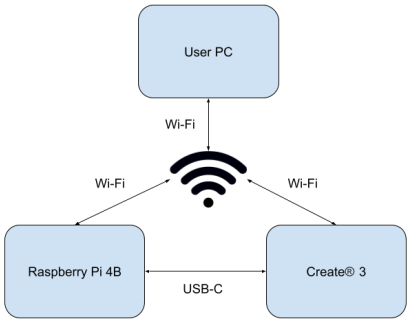

Figure 1: TurtleBot 4 computer connections

The connection between TurtleBot 4 and user PC should be configured and set up. The connections between the two ends are shown in figure 1. Create 3 and Raspberry Pi are located inside the TurtleBot 4 and connected together via a USB. The three nodes should be connected to the same WiFi to allow communications. This procedure is done by following a tutorial presented in [1].

2.2. ORB-SLAM2

Simultaneous Localization and mapping (SLAM) techniques is used to build a map of an unknown environment and then localize the camera sensor in that map [9]. SLAM focuses on real time operations and provides accurate place recognition. This is important to detect when the sensor returns to an area that is already mapped and can correct errors in exploration (close loops). ORB-SLAM2 system has three main threads: Tracking, local mapping and loop closing (shown in figure 2). Tracking is used to localize the camera in the local map, this is done by finding features that matches the local map and then apply motion-only bundle adjustment(BA) to the re projection error. Local mapping manages the local map and does optimizations on it by performing local BA. Loop closing detects large loops and corrects the accumulated drift, this is done by preforming a pose-graph optimization. The loop closing thread also lunches a fourth thread that perform a full BA, this is done to compute the optimal structure and motion solution.

2.3. Nav2

Nav2:s navigation architecture is based on a behaviour tree [5] see figure 3 with a logic of left to right. It has a policy that ticks a global planner action at a rate of 1 Hz. It defaults to trying to find a path using an A* and planner combination, if it were to fail it will fall back in the tree and take

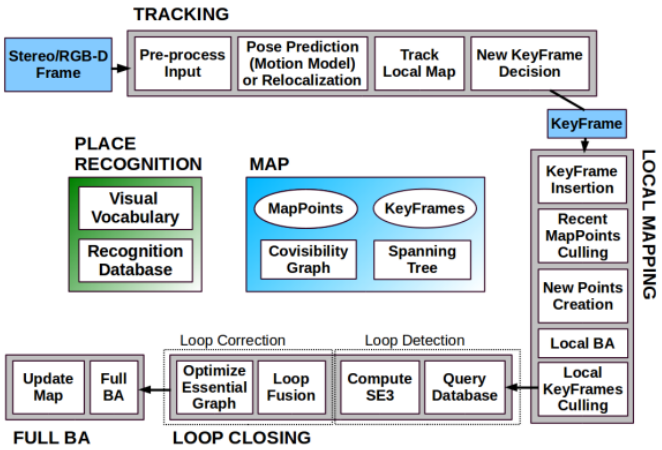

Figure 2: System Threads and Modules.

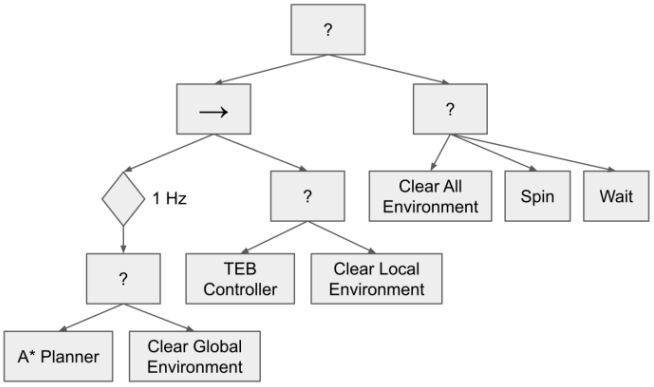

Figure 3: Nav2:s Behaviour tree, where "?" represents a fallback node and "→" a sequence node

the next child node where it cleared the global environment node, this can help resolve potential failures. If that fails it will traverse up the tree and check the next child node.

To ensure safety for robots moving near humans Nav2 is built on top of ROS2 to address functional safety standards. This component will be used in the project for navigating the environment.

2.4. YOLOv7

The YOLO [4] architecture is based on a fully connected Neural Network approach. YOLO:s framework has three core components the backbone, neck, and head. The backbone in YOLOv7 [12] is trained on the COCO dataset. The backbone is set up whit an Extended Efficient Layer Aggregation Network which enables YOLOv7 to enhance the learning ability of the network. This layer is also one of the core components in making YOLOv7:s bounding box predictions a lot more accurate than previous versions. The head of architecture is responsible for making the prediction. YOLOv7 is not limited to only one head and is composed of multiple heads, where a lead head is responsible for the final prediction. The other auxiliary heads are used to assist in the training of the middle layers. The weights of the auxiliary head are changed based on the assistant loss. The lead head and the label assigner are later combined to make the final prediction. In simple terms, the label assigner generates soft labels.

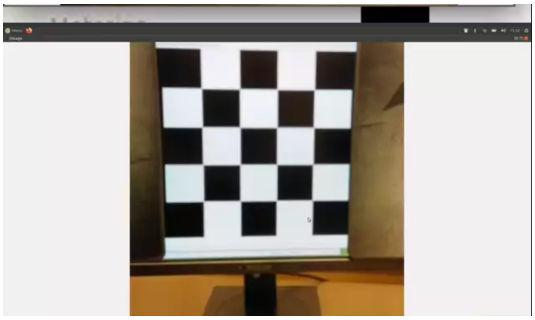

Figure 4: Picture taken by Turtlebot for the process of calibrating the camera

2.5. Object to World Position

Once the YOLOv7 model has computed the label, bounding box, and prediction score, the data is used to translate it to the real-world position of the object (global position). The first step of this process is to calculate the intrinsic matrix of the camera using the OpenCV calibration method. This method uses an image of a checkerboard pattern taken by the camera to measure the parameters of the lens and sensor, see figure 4. These parameters are then placed into the intrinsic matrix and can then be re-used for all further calculations.

$$intrinsic\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 0 \end{bmatrix} \quad (1)$$

The second step of the process is to create a direction vector of the point of interest. This is done by using the focal length and principal point from the intrinsic matrix:

$$d_x = \frac{x - c_x}{f_x} \quad (2)$$

$$d_y = \frac{y - c_y}{f_y} \quad (3)$$

$$d_z = 1 \quad (4)$$

The direction of the target can now be scaled by the distance to the target, sampled at (x, y), to get a relative position of the object. This process is illustrated in figure 5.

The relative position, together with the data from ORB-SLAM2, can be converted to a global position. This is

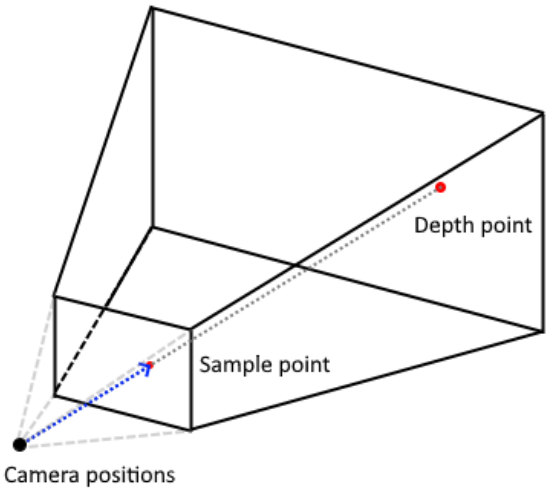

Figure 5: Conversion between 2D point to 3D point at sampled depth

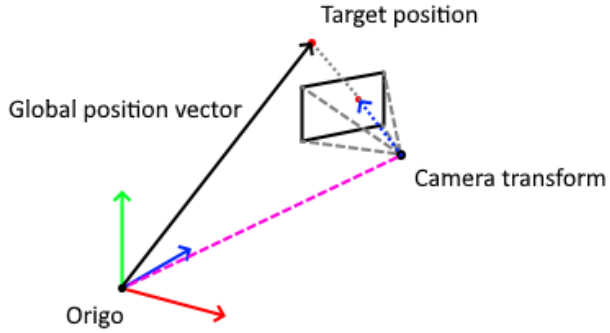

Figure 6: Visualisation of converting between relative and global positions

done by first applying the robot's rotation, given by ORB-SLAM2, to the relative position and then adding the robot's position. The result is a vector representing the position of the object in world space, independently from the robot. This process is illustrated in figure 6.

The width of an object can be calculated using a technique similar to the position:

$$width = \frac{x_{right} - x_{left}}{f_x} depth \tag{5}$$

2.6. Bounding Box Volume

When an image bounding box has been computed, the size is given in pixel units. To make this information useful, it must first be converted to world space units using the camera calibration data. This gives an image bounding box in

Figure 7: Visualisation of the bounding box volume

metric units that can be used to estimate the volume of the object. In order to simplify tracking the volume of objects, it is assumed that all objects are cylindrical. This allows for easy tracking as the width of the image bounding box always corresponds to the diameter of the cylinder. This creates a planar-symmetrical shape representation of all objects that is easy to update and boundary check. A visualisation of the bounding box volume can be seen in figure 7.

2.7. Bookkeeping

The bookkeeping system keep tracks of detected objects. It stores 3D positions, bounding box volume, labels and timestamps of all objects.

Each processed frame computes candidate objects represented by a 3D position, a bounding box volume and a label. These are then checked against the stored objects to determine if the object has been seen before. This check is done by determining if the following conditions are true:

1. The labels match
2. The centre position is within another objects cylindrical bounds

If these conditions are met the system assumes the objects are the same and update the existing object's timestamp, average the positions and sets the diameter of the cylindrical bounds to the maximum width. If the conditions are not met the system assumes that it's a new object or a potentially moved object and is inserted as a new entry. Only after the Turtlebot has patrolled the entire environment can an actual check be performed to see if one object of this kind has actually been added or changed position. This can be determined by checking the list of registered objects from the previous scan against the current. If there is, in total, more new objects, of a specific label, than remove objects the system assumes that the object it is a new addition. However, if the total number of object, of that label, are equal, the object is assumed to have been moved.

To minimize the number of dynamic objects that could cause noise in our system a blacklist of certain candidate objects is checked before they are processed. This is done by

checking the computed label of the object against a blacklist of names that are dynamic objects. Examples of this would be: a person or a dog.

To determine if an object no longer exists the system uses the list of timestamps. If a timestamp is older than when the system first started the scan it can assume that the object has been moved and will prune it from the system.

This system has a drawback when objects of a specified label have both been moved and added to the environment. For example, if there was only one object in the environment and the next time there are now 2 objects of the same type and both are in a new location it can never be known which object has been moved and what has been added. Therefore, this will be based on which order the robot found the objects.

3. Problems

3.1. Laboratory and Turtlebot access

The group spent the first three weeks of the project researching and waiting to gain access to the laboratory. Once they finally gained access, they found that the lab was set up in such a way that they needed to change passwords every day, and each computer had to have a different password. This made it difficult to remember which passwords belonged to which computer, as they shared the space with multiple students, and the computers were only sometimes available. Some group members were limited to using only one or two computers because of the frequent password changes. After another two weeks, this problem was finally resolved, but every computer retained the last password set on it.

3.2. Turtlebot Network Connection

For this project, the group used Turtlebot 4 to collect data for their pipeline. They gained access to the university's AI lab computers, which were set up in containerized Docker environments. The Turtlebot 4 was configured to send its data as a UDP multicast. The group spent the following three weeks debugging the Turtlebot with the help of Liu network technicians. However, the technicians eventually stopped responding to the group because they did not have enough time to provide further assistance.

3.3. The Laptops

The group's customer, Trafikverket, was kind enough to purchase a powerful laptop for the Turtlebot 4. This laptop was finally configured the week after the first examination period. However, the wrong version of Linux was installed, and the drivers for the network card were not working. This meant the laptop could either be connected to the internet or connected via cable to the Turtlebot, but not both at the same time. This made working with the laptop very difficult. The group had root access to the laptop but still had to use Docker to install the correct version of Linux. At this point, they realized that UDP multicasting was not supported in Docker. Fortunately, a Liu technician was able to help them install the correct version of Linux and fix the network driver issues, allowing them to at least listen to the topics that the Turtlebot 4 was sending. However, this was done on December 8th, and the project was due on December 12th, so the group needed more time to explore this new opportunity.

3.4. Ros2 Bag

To be able to use any data at all, the group was advised to record ROS2 bags. One group member managed to install the correct version of Linux on his personal laptop. From there, they planned to record data from the Turtlebot 4 as it drove around the lab. They managed to record a 2-minute bag using this laptop, but when they later tried to read the bag, they discovered that it did not contain any image data. After they figured out how to properly include image topics, they tried to record again, but the queue filled up, and no data was collected. The group now believe that the network connection and the personal laptop need to be stronger to handle all the data.

3.5. Hotspot

Throughout the project, the group was promised a dedicated hotspot in the form of a router. While they waited for this, they used their mobile phones and computers as hotspots. However, once they realized that the computer they were using for bagging could not receive the data fast enough, they concluded that the hotspots needed to be faster to keep up with sending the data. Unfortunately, they never received a more powerful hotspot and were unable to collect any data. Additionally, the Turtlebot was meant to be used with a 5Ghz network connection. Since our hotspots were only 2.5 GHz, this could have put additional limitations, which could have contributed to the problem.

3.6. Data Collection Using Ros2bags

As previously mentioned, the group could not obtain any useful data from the Turtlebot, so they decided to resort to Plan B, which was to download pre-recorded ROS2 bags. However, the open-source bags they found were not from a robot similar to the Turtlebot 4. They also discovered that the bags only contained raw sensor data that had yet to be processed through ORB-SLAM2 and Nav2. The bag they found featured a tall robot, unlike the Turtlebot, which drives on the floor. This robot could also turn its camera up, down, left, and right. As a result, the static calculations for the direction and angle to determine where it was looking were not as straightforward as for the Turtlebot. This also made it nearly impossible to test their bookkeeping system, as the bagged data they found only went around their lab

Figure 8: Object detection and localization performed on the third-party image using YOLOv7.

once. Therefore, they never had a second look at the different rooms and could not build up their bookkeeping system to do anything useful.

3.7. Turtlebot Simulator (Gazebo)

The group could not use the Turtlebot to obtain any useful data; thus they decided to try out the simulator. After booting up the simulator for the first time, the group noticed it was running very slowly. They determined that this was because the docker container did not have GPU access. Because the group was forced to still work within a Docker container, it wasn't easy to achieve GPU passthrough, and the group could not find a working solution. During the research, they also realized that even if the simulator had worked, the data that would have been acquired would only have allowed them to implement the robot's initial scanning and patrolling aspects. The environment in the simulator was a 3D model with very few objects, and the YOLOv7 model that was used was not trained on 3D-modelled boxes and shelves, so the likelihood of detecting any objects was very low. Given these drawbacks, it was decided to abandon this approach to have more time to work on the Turtlebot.

4. Results

4.1. Mapping Environment for Localisation

ORB-SLAM2 and Nav2 ended up working well for the given situation. The resulting ability to generate a map from the environment of the Turtlebot was deemed satisfactory. Given that SLAM can operate in real-time (as mentioned in 2.2), the implemented algorithm could be used in both simulations and live in the robot with good performance. Using the ROS-package RViz, the produced map could also be visualized to see what parts of the environment have been mapped out, the current location of the robot etc.

4.2. Navigating the Environment

Because of the ability to map out the surrounding environment, the possibility to also navigate the given environment was enabled. While navigation was possible, there were some slight issues with its usability. To begin with, the Turtlebot does not feature any user interface, so the capability of navigating the environment with the robot could have gone through more thorough testing. This navigation for the robot was possible with both a keyboard and waypoints. However, using the keyboard often got slow, and trying to use waypoints often did not work. The reason for it not completely working properly may be because of connectivity issues, as well as the laptop that was used possibly needing to be faster. This is because the queue kept filling up, which kept causing out-of-sync errors, this is not fully confirmed, however, and it is only what is believed to be the reason for it not completely working. In the simulation, though, the navigation yielded much more promising results. The robot was able to move to different positions in a previously mapped environment when it was given the orders to do so; it was able to do so while also avoiding existing objects and obstacles.

4.3. Detecting/Classifying Objects

When integrating the YOLOv7 library into the project, one challenge was that a direct function needed to process a single image simultaneously. To overcome this, specific code from the YOLOv7 library relevant to our needs was modified and adapted to fit the implementation.

To test the modified code from YOLOv7, the Turtlebot 4 camera and third-party image dataset described in [7] were used. Most testing was done using third-party image datasets because of the problem described in section 3.6. Besides, it was essential to evaluate the implementation's performance in various environments. The testing data allowed the group to confirm that the modified code was functioning correctly.

In Figure 8, YOLOv7 are tested on the third-party image. A chair and TV are detected, but there are other objects in the picture that the YOLOv7 model could not detect. This may happen since the fastest version of weights is used in YOLOv7, which is 2.8 ms/image 10. The use of the fastest version was required to speed up the detection process and allow time for data transfer between the PC and the Turtlebot 4 and other required calculations. That is to say, a trade-off for increased speed is that the accuracy of the object detection may be reduced, resulting in some objects being missed. This version allowed real-time processing of the frames at 60 frames per second. Another reason why YOLOv7 may not have detected some objects is that the model provides a confidence level for each object detection, indicating how confident it is in its prediction. If the confidence level for a particular object is below the threshold (set

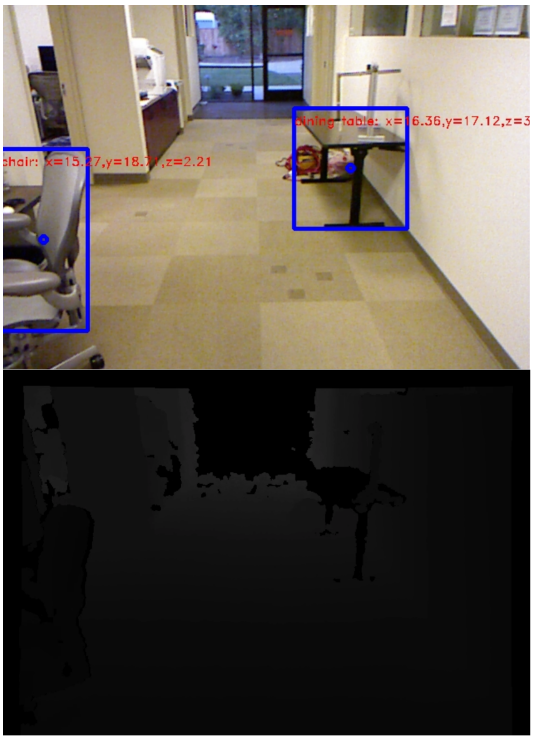

Figure 9: Depth map applied to third party image. White pixels indicate that the corresponding pixels in the image are farther from the camera, while dark pixels are closer to the camera.

by the user, in this case, 0.25), the object is removed from the detection results.

After detecting the objects in an image, the local position of each object is determined relative to the Turtlebot or the location from where the picture was taken. This is achieved by calculating the centre of the bounding box of each detected object and applying a depth map, as shown in Figure 9. This information, together with the camera's intrinsic parameters, is used to estimate the local three-dimensional position of the detected objects.

When the local position has been calculated, the global position can be retrieved by adding the extrinsic properties of the camera. In our case, this is the transform position and rotation. Due to the limitations described in problem 3.6 3.4, part of the system was not accurately tested. However, using rough estimations, it is observed that the system seemed to work correctly.

In Figure 9, it is observed that there is a limit to the depth map. Objects that are too far away or too close become dark, 0 value, since the depth camera cannot estimate depth over or under a certain distance. This indicates that the results of the depth map are not always accurate. Thus, any values exactly 0 are ignored since they can not be trusted.

Accurate calculation of the local position of objects in our system required a reference point on the object for use as a basis for depth calculations. However, using the centre of an object may only sometimes be possible, as some objects, such as tables, are not solid. In these cases, depth measurement would be calculated from a point behind the object and thus lead to an incorrect object position.

To address this issue, several different methods were considered. These included randomly selecting points within the bounding box and taking the average, but this approach is still affected by the non-solidity of certain objects; taking a diagonal line across the bounding box and averaging the depth values along that line, which also suffers from the same issue; and selecting the closest point of the object to the camera, which does not account for the possibility of other objects being in front of the object in question.

As a final solution, a blacklist approach were implemented in which objects with labels indicating that they were not solid were excluded, such as chairs, tables, and people. For the remaining solid objects, centre position is used as the depth value.

One problem was removing lens distortion from the robot's camera images. To do this, it was necessary to find the camera's intrinsic parameters, including its focal length and optical centres. Images with well-defined patterns are used 4, such as a chessboard. Specific points on the pattern were identified for which the relative positions were known and used to solve for the distortion coefficients. Every chess square on the board was 5,5cm X 6,0cm. That was tested by taking a picture of a ruler to check that the calibration was working. As a result, the size of objects was successfully estimated with an accuracy of +/- 0.2cm.

4.4. Bookkeeping

Only a basic bookkeeping system was ultimately implemented due to several factors. Firstly, time constraints arose from previously encountered hardware issues. In addition, the reliance on bagged data, which did not contain the necessary timestamps for effective bookkeeping, also contributed to the limited implementation. Furthermore, the bagged data did not have the robot's transformation, resulting in an inability to determine the view angle of the robot automatically. As a result, the accurate positioning of objects in the environment could not be determined. As if the previously mentioned problems were not enough, the robot from the bagged data never revisited any of the locations that it had already been. This meant that the testing of an eventual bookkeeping system would have been impossible.

4.5. Patrol the Environment and Scan for Objects

The implementation of patrolling was not completed due to the previously mentioned inability to connect the provided laptops to the Turtlebot. As a result, only the initial patrol was carried out. The initial patrol was intended to map

Model	Test Size	APtest	AP50test	AP75test	batch 1 fps	batch 32 average time
YOLOv7	640	51.4%	69.7%	55.9%	161 fps	2.8 ms
YOLOv7-X	640	53.1%	71.2%	57.8%	114 fps	4.3 ms
YOLOv7-W6	1280	54.9%	72.6%	60.1%	84 fps	7.6 ms
YOLOv7-E6	1280	56.0%	73.5%	61.2%	56 fps	12.3 ms
YOLOv7-D6	1280	56.6%	74.0%	61.8%	44 fps	15.0 ms
YOLOv7-E6E	1280	56.8%	74.4%	62.1%	36 fps	18.7 ms

Figure 10: Performance of different models of YOLOv7 [3].

the environment and detect objects, with subsequent patrols meant to monitor any detected objects. However, the inability to establish a connection with the Turtlebot hindered the full realization of this plan.

4.6. Generate a Textured Mesh of the 3D Environment

Generating a 3D representation of the environment was one of the requested features. Our intended approach to solving this problem was to use a library called Voxblox [10] since the existing techniques used, such as ORB-SLAM2, would contain the information required to create a sufficiently accurate textured environment mesh. However, this was unfortunately not implemented since Voxblox required a properly localized transform with a functioning camera and depth stream.

4.7. Simulator

Parallel to work on the robot, there was an effort to get the simulator to work. Gazeebo was the simulator that was tried to be used. However, some problems getting the simulator to work properly never got fixed. These issues were:

1. GPU pass-through
2. Rvis integration

The first problem mentioned was GPU pass-through. Because GPU pass-through never worked properly, the simulator ran extremely slow, making it extremely unpleasant to work with. This is because of the time actions took to perform. The actions performed usually took a few seconds to register and made the program feel very unresponsive. The second issue with the simulator was the connection to Rvis. The integration with Rvis and Gazeebo did not work as intended. More precisely the simulator provided no usable sensor data that could be used for further work. Because of the issues listed, the decision was made to put more energy into getting the actual robot to work. This is because, at the time, it seemed to be the best alternative to get working results. As the robot gave sensor data, such as camera footage, that could be used in, and known to work with, the YOLOv7 implementation.

5. Discussion

5.1. Mapping Environment for Localistion

The mapping functionality worked fine and would not necessarily need a ton of extra work; the current implementation would not benefit a lot from having more time spent on it, as it did not seem to have many issues and had a good performance.

5.2. Navigating the Environment

Given that one of the key components of the project is about the robot's ability to navigate around in its environment, as a stationary robot would not be very useful for the desired purposes, it is certainly not the greatest situation that the navigation could not go through much testing. The non-ideal conditions that had to be worked in (i.e. the previously mentioned connectivity issues and inability to make use of the provided laptop and computers) made progressing with this task difficult and, to an extent, hindered progress on other tasks that built upon the navigation ability, such as patrolling the environment and scanning for objects.

5.3. Detecting/Classifying Objects

Because major testing on third-party images was conducted, some disadvantages could be improved upon. One disadvantage is that there was limited control over the objects that were used in the dataset. This will limit the project to testing on specific types of objects, which may not satisfy the project's needs and limit the objects' diversity.

Detecting the object and giving it the correct label is essential in this work, which is needed to be improved in this conducted work. One solution to improve this is to use a better version of the model in YOLOv7, which may require additional computational resources. Access to more powerful computing resources and a faster router between the bot and the computer can enable this improvement.

While the blacklist approach implemented as the final solution represents a significant improvement over the previous methods in calculating the local position of solid objects, it has some limitations.

While the centre position approach is effective for solid objects, it is also sensitive to the possibility of other objects being in front of the object in question. This could lead to incorrect depth calculations for objects that are partially occluded. To address this issue, one possible improvement would be to use machine learning techniques to train

a model to identify appropriate points to calculate the depth on.

5.4. Bookkeeping

The implementation of the bookkeeping system highlighted the importance of having accurate and comprehensive data when developing such a system. In this case, the lack of timestamps and transformation information made it difficult to accurately track the movements and actions of the robot, which is crucial for effective bookkeeping. It could also be noted that the inability to test the system due to the lack of revisited locations hindered the development and evaluation of the bookkeeping system.

One potential way to address these issues in the future could be to prioritize the collection of more complete and accurate data, including timestamps and transformation information, when developing a bookkeeping system for a robot. However, as previously mentioned, the project was plagued with non-working hardware which was the underlying cause to all these problems.

5.5. Patrol the environment and scan for objects

The inability to connect the laptops to the Turtlebot had significant implications for the successful implementation of the patrolling task. While the initial patrol was able to map the environment and detect objects, subsequent patrols were not able to be carried out as intended. This was a limitation of the project, as monitoring detected objects was a key component of the patrol plan. This limitation highlights the importance of ensuring proper hardware setup in order to successfully carry out robotics tasks.

5.6. Generate a Textured Mesh of the 3D Environment

Although this feature was not implemented, some potential issues were identified One of the intended uses of this mesh was virtual reality (VR), but the mesh itself could potentially require additional processing to be suitable for such an application. Due to the nature of how the data is collected, Voxblox produces a mesh with holes in it. This is caused by the limited amount of camera angles that a small ground-based robot is able to capture. To improve the results, it would be beneficial to fill in the holes and make the mesh more consistent with the environment. During the research, no sufficient method for processing the mesh was found. One potential approach would have been to run an edge-loop detection algorithm that simply created new faces for any edges of the mesh that did not already belong to a closed face. This would create a closed mesh but would also create artefact volumes behind objects that from some angles would look odd and inaccurate.

Using Voxbox would also limit the environment from containing any dynamic objects during capture since it would create clear artefacts. In hindsight, a better approach to explore would perhaps have been point cloud generation, which is built up over time and become denser. This would resolve the issue for fast-moving dynamic objects since they would, hopefully, become noise.

Near the end of the project, another potential method for creating a 3D environment representation was discovered, neural radiance fields (NeRf)[8]. This is a rather new way of representing an environment that is based on over-fitting a neural network on the camera perspective images. The initial proposal for NeRFs were based on the assumption that the scene is static and would likely have similar effects to that observed by Voxblox. However, a more recent study has shown that it is possible to create a NeRF that accounts for dynamic changes[11]. This would have been interesting to explore more since NeRFs have a clear advantage in visual fidelity over traditional techniques since it is able to maintain both reflection and refraction. NeRFs are also quite small in comparison to complex meshes and NVIDIA has even shown that it is possible to make this a dynamic property through variable bitrate [13].

5.7. Simulator

As mentioned many times in this project not everything went as planned, this also reflects the effort when trying to get the simulator to work. Many of the problems encountered were partially due to the requirement to use a Docker environment. GPU pass-through would not have been an issue if it were possible to work outside the Docker environment. The issues faced were mainly due to the use of the Docker environment, but it was the only alternative available because the operating system requirements for running ROS2 Galactic on the Turtlebots were different from the operating systems on the available computers. Running the simulator on personal computers also had issues, mainly related to windows. Potential improvements to the situation would have been to run the simulation on a machine that had the correct operating system and not in a docker container. This would remove both the issue with GPU pass-through, though, in this situation, the problems with the physical robot would also be solved. But being able to work in the simulation would have clear benefits like being able to test code without needing to deal with the robot. There is also the benefit that more group members could work with the robot at the same time, something that is hard when you only have one robot.

6. Conclusions

With an initial anticipation and excitement for the project the group unfortunately leave this project with a feeling of wanting to do more. To have to spend hours upon hours researching why our specifically configured environments dose not work was not the idea of the project we had an-

ticipated. However, this project did teach us how to focus on what was important and to react and adapt to situations that were unforeseen. This is of course the reality for many projects but in this case, it was more so than what we had previously experienced.

References

[1] Turtlebot 4 - quick start, 2022.

[2] Z. T. A. M. Helen Oleynikova, Marius Fehr and others. Real-Time SLAM for Monocular, Stereo and RGB-D Cameras, with Loop Detection and Relocalization Capabilities, 2018.

[3] W. Kin-Yiu. yolov7. `https://github.com/WongKinYiu/yolov7`, 2022.

[4] S. R. Kukil. Yolov7 object detection paper explanation and inference. 2022.

[5] S. Macenski, F. Martin, R. White, and J. G. Clavero. The marathon 2: A navigation system. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, oct 2020.

[6] S. Macenski, F. Martin, R. White, and J. Ginés Clavero. The marathon 2: A navigation system. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[7] J. Mason and B. Marthi. An object-based semantic world model for long-term change detection and semantic querying. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3851–3858. IEEE, 2012.

[8] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *CoRR*, abs/2003.08934, 2020.

[9] R. Mur-Artal and J. D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*, 33(5):1255–1262, 2017.

[10] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto. A library for flexible voxel-based mapping, mainly focusing on truncated and Euclidean signed distance fields, 2022.

[11] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10318–10327, 2021.

[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection, 2015.

[13] T. Takikawa, A. Evans, J. Tremblay, T. Müller, M. McGuire, A. Jacobson, and S. Fidler. Variable bitrate neural fields. In *ACM SIGGRAPH 2022 Conference Proceedings*, SIGGRAPH '22, New York, NY, USA, 2022. Association for Computing Machinery.