Cleaning Robot LuminaAI Project: Real-Time Learning for Smart Home Cleaning

Angel Ivan Mayo
Computational Robotics Engineering
Universidad Politecnica de Yucatán
Km. 4.5. Carretera Mérida — Tetiz
Tablaje Catastral 4448. CP 97357
Ucú, Yucatán. México
Email: 2009093@upy.edu.mx

Ricardo Hernández Ramírez
Computational Robotics Engineering
Universidad Politecnica de Yucatán
Km. 4.5. Carretera Mérida — Tetiz
Tablaje Catastral 4448. CP 97357
Ucú, Yucatán. México
Email: 2009070@upy.edu.mx

Braulio Millán Chin
Computational Robotics Engineering
Universidad Politecnica de Yucatán
Km. 4.5. Carretera Mérida — Tetiz
Tablaje Catastral 4448. CP 97357
Ucú, Yucatán. México
Email: 2009095@upy.edu.mx

Axel Esdras Flores
Computational Robotics Engineering
Universidad Politecnica de Yucatán
Km. 4.5. Carretera Mérida — Tetiz
Tablaje Catastral 4448. CP 97357
Ucú, Yucatán. México
Email: 1909065@upy.edu.mx

***Abstract*—This document outlines the development and implementation of a project focused on integrating various machine learning models into a home cleaning robot. The project aims to enhance the robot's capabilities in object and obstacle recognition, autonomous mapping and navigation, adaptation to environmental changes, and intelligent real-time decision-making for improved cleaning efficiency. It encompasses three main learning models: supervised, unsupervised, and semi-supervised. In the supervised learning segment, a TensorFlow-based neural network is designed to recognize household objects like chairs and doors, utilizing techniques like Conv2D and MaxPooling2D. The unsupervised learning part employs an Isolation Forest algorithm to detect anomalies in robotic arm kinematics, a crucial aspect for autonomous robot functioning. Lastly, the semi-supervised component uses the Catkin Workspace and TurtleBot3 simulation space for developing a SLAM GMAPPING model, enabling the robot to map its environment and optimize cleaning routes.**

***Index Terms*—Autonomous robotics, Real-time machine learning, Deep learning algorithms, Graphical User Interface (GUI), LED lighting effects, Neural networks, Obstacle detection, Adaptation in real time, Real-time visualization, Cleaning efficiency.**

1. Introduction

Home cleaning robots are one of the most common domestically used robots of the current period. These robots make use of machine learning algorithms to map the room they are currently in, identify and avoid obstacles, and can determine the most optimal path to conserve energy and cleaning efficiency. For this project, we decided to tackle some of the challenges of designing machine learning models for a home cleaning robot.

There are three types of machine learning models we had to design for the robot: A supervised model for an object identification algorithm that the robot will use through its cameras to detect and classify objects in front of it. An unsupervised model where, through the use of the Isolation Forest algorithm, the cleaning robot can identify anomalies in its functioning. And finally, a semi-supervised model where through the use of gmapping and Simultaneous Localization and Mapping, the robot will identify the optimal path.

2. Objectives

2.1. General objectives.

- **General Objective 1 (Supervised). Object and Obstacle Recognition** Implement a supervised learning system that allows Cleaning Robot LuminaAI to recognize and classify common objects and obstacles in a household environment, such as furniture, toys, and doors.
- **General Objective 2 (Unsupervised). Autonomous Mapping and Navigation** Develop unsupervised learning algorithms for Cleaning Robot LuminaAI to construct detailed maps of its environment and autonomously navigate, identifying cleaned areas and optimizing its route in real-time.
- **General Objective 3 (Unsupervised). Adaptation to Changes in the Home** Train Cleaning Robot LuminaAI to use unsupervised learning, enabling

it to dynamically adapt to changes in the home environment, such as furniture rearrangements or the presence of new obstacles.

- **General Objective 4 (Reinforcement). Intelligent Real-Time Decision Making** Implement a reinforcement learning system that enables Cleaning Robot LuminaAI to make intelligent real-time decisions to optimize cleaning efficiency and navigation, considering feedback from the environment and cleaning goals.
- **General Objective 5 (Reinforcement). Continuous Performance Improvement** Utilize reinforcement learning for Cleaning Robot LuminaAI to continually improve its performance over time, adapting to changes in the home and refining its cleaning and navigation strategies based on past experiences.

2.2. Specific objectives.

- **Supervised Learning to develop an Object Recognition Model** Design, train, and validate a supervised learning model that enables Cleaning Robot LuminaAI to identify and classify common objects and obstacles present in a household environment.
- **Unsupervised Learning for an autonomous Map Creation** Implement unsupervised learning algorithms that allow Cleaning Robot LuminaAI to autonomously construct and maintain detailed maps of its environment. The robot should achieve the capability to generate accurate maps with a spatial resolution of at least 5 cm.
- **Reinforcement Learning for a Cleaning Route Optimization** Develop a reinforcement learning system that enables Cleaning Robot LuminaAI to optimize its cleaning routes in real-time. The robot should learn to minimize cleaning time and efficiently navigate around obstacles, gradually improving its performance over time.

These specific objectives align with the general objectives and represent clear and achievable goals for addressing each of the aspects of supervised, unsupervised, and reinforcement learning in your project.

3. THEORETICAL FRAMEWORK

3.1. Fundamentals of Machine Learning in Robotics

The success of advanced robotics projects, such as Cleaning Robot LuminaAI, relies on a solid understanding of the fundamental principles of machine learning applied to the field of robotics. This section of the theoretical framework establishes an essential conceptual foundation.

3.1.1. Autonomous Robotics and Machine Learning. Aims to develop robots capable of performing tasks without constant human intervention. Machine learning plays a crucial role by enabling robots to acquire knowledge and make autonomous decisions based on data and past experiences.

3.1.2. Types of Machine Learning in Robotics.

- Supervised Learning: In this approach, labeled training data is provided to the robot to learn specific tasks, such as object recognition or decision-making.
- Unsupervised Learning: Here, the robot seeks patterns and structures in data without the guidance of labels, which can be useful for mapping and adaptation in unfamiliar environments.
- Reinforcement Learning: In this type of learning, the robot makes decisions in its environment and receives rewards or penalties based on its actions. This is fundamental for autonomous decision-making.

3.1.3. Sensors and Actuators in Robotics. Robots use sensors to perceive their environment, such as cameras, ultrasound sensors, and lasers. Additionally, they have actuators to perform physical actions, such as motors and wheels. The combination of sensors and actuators allows robots to interact with the external world.

3.1.4. Mapping and Localization. Are critical aspects of autonomous robotics. Mapping involves creating representations of the surrounding environment, while localization involves determining the robot's position in that map. These processes rely on machine learning techniques to achieve accuracy and robustness.

3.1.5. Path Planning. Involves generating optimal paths for the robot to reach its goals efficiently while avoiding obstacles. Machine learning can enhance the robot's ability to plan adaptive routes in real-time.

3.2. Machine Learning Models and Algorithms

In this section, we delve deeper into the machine learning models and algorithms relevant to robotics and their applications:

3.3. Neural Networks and Deep Learning

They have become pivotal in various robotics applications. Convolutional neural networks (CNNs) excel in image recognition tasks, making them valuable for object recognition by robots. Recurrent neural networks (RNNs) and Long Short-Term Memory (LSTM) networks aid in sequential data analysis and decision-making.

3.4. Simultaneous Localization and Mapping (SLAM)

It is a fundamental technique for robots to map their surroundings while simultaneously determining their own position within the map. Machine learning plays a role in

enhancing SLAM accuracy and robustness, enabling robots to navigate in complex and dynamic environments. SLAM is specially useful on home cleaning robots because it allows the robot to map the room it is currently in and use its capabilities efficiently, reducing time and battery consumed.

3.5. Reinforcement Learning Algorithms

Algorithms like Q-learning and deep reinforcement learning (DRL) have been instrumental in training robots to make optimal decisions. DRL, in particular, has been applied to teach robots complex tasks, such as robotic manipulation and control.

4. Methods and Tools

4.1. Methods

4.1.1. Supervised. for this type of model we use a model inside TensorFlow API called sequential. This premade model can be transformed into a neural network capable of detecting objects inside images. We also use the Conv2D, MaxPooling2D, Dense, Flatten libraries to create the model and its behaviour.

4.1.2. Unsupervised. for this task we developed an Isolation-Forest from scratch only using numpy, pandas and other libraries for file directories.

4.1.3. Semi-Supervised. In building our SLAM GMAPPING model, we used Catkin Workspace, part of the Robot Operating System (ROS), to organize and manage our code, done through Linux on Ubuntu 20.04 as our operating system since it's a dependency for Catkin and ROS.

We relied on the TurtleBot3, a ready-made robot. This saved us time because we didn't have to build a robot from scratch, even when it came to simulating the movement. Instead, we could focus on perfecting our SLAM GMAPPING model and making sure it worked seamlessly with the bot.

4.2. Tools

Computer, google drive credentials, google colab, Opera extension Download All Images. Python libraries: Tensorflow, Scikitlearn, os, OpenCV, numpy, matplotlib. Catkin Workspace, ROS Noetic, Ubuntu 20.04, TurtleBot3 Simulation space.

5. Development

5.1. Supervised method

Supervised machine learning models are trained using a labeled dataset. Each input data point, there is a corresponding correct or "ground truth" output. The goal of supervised learning is to learn a mapping function from the input data to the output labels, such that the model can make accurate predictions on new, unseen data.

5.1.1. Data Collection. For the first process an extension for web browsers is used, Download All Images. This program downloads every picture of every format that the browser loads, thanks to the tool, a large dataset was obtained easily and without much complications. A simple method is also apllied to cleanse more the images, every picture of less than 15 kilobytes is deleted as it would be very small for the model.

The next step for data collection is the loading of data. Images are loaded into a google drive, colab has the capacity to link drive spaces to the jupyter environemnt. By linking the drive account with the colab the images are loaded for the model without having to use local memory or processing.

5.1.2. Data Preprocessing. The process involves several steps due to the complexity of creating neural networks. The first step of image preprocessing is to delete images that are corrupted or not pressent inside the loaded folder, this avoids errors when training. A function is declared for this process, it has a for loop that will compare the cv2 reading with the image integrity, if the imformation does not coincide, the images is deleted.

```
for image_class in os.listdir(data_dir):
    for image in os.listdir(os.path.join(data_dir, image_class)):
        image_path = os.path.join(data_dir, image_class, image)
        try:
            img = cv2.imread(image_path)
            tip = imghdr.what(image_path)
            if tip not in image_exts:
                print('Image not in ext list {}'.format(image_path))
                os.remove(image_path)
        except Exception as e:
            print('Issue with image {}'.format(image_path))
```

Figure 1. Code used for removing corrupted images.

Next is the loading of data into the keras pipeline. Using a pipeline is really usefull for heavy datasets like image ones. With keras we can define the type of information, images for the case, and create an iterator to later create a batch and train the model more efficiently.

Everytime the iterator runs, the batch changes. Batches can be displayed and configures to meet the requirements, but for this project the standard configuration is used. Keras on his own assimilates which one will be 0 or 1, for this model in particular it choosed to be 0 = chair, and 1 = door. The final key step is to resize the batches pixels so they consume less.

Lambda is used to define an anonymous or inline function in Python. The lambda function is used to map the function and perform an operation on every element of the data collection.

5.1.3. Data Split. Machine learning models needs training an testing data. With images is the same, the best ratio is to have 80 percent as training images and the remaining 20 percent as testing images. Remember that the images have been divided into batches, len(data) is used to see the

amount of batches the pipeline has. These are divided into train, validation, and test sections. Because the information is impair, an additional integer is added to pair them and complete the amount of total batches. Functions are used to tell the pipeline to not repeat images inside other partitions of the data.

5.1.4. Contruction of the Model. Sequential model from the TensorFlow API is used, and predefined layers Conv2D, MaxPooling2D, Dense, and Flatten are loaded. Two activation functions are used, relu function is used for the majority while sigmoid is only used on the last layer. The neural network is comformed by an initial layer of 16 neurons, then 32, then 16, then one dense layer of 256 neurons, and finally a dense layer of 1 neuron.

The Adam optimizer, short for Adaptive Moment Estimation, is used because it can compute the gradient of the loss with respect to model parameters, update the exponential moving average of past gradient, update the exponential moving average of past squared gradients, use these moving averages to adaptively compute the learning rates for each parameter, apply a scaled version of the gradient to each parameter to combine both the momentum and adaptive learning rates.

tf.losses.BinaryCrossentropy sets the loss function that the model will use during training. This loss function is commonly used for binary classification tasks, 1 or 0. It measures the dissimilarity between the predicted values and the true labels.

TensorFlow allow us to know many parameters of the model. Accuracy is information needed to understand how effective the model is, it is based on the evaluation of the testing set with the model decisions.

5.1.5. Training. Before training the model, a log is created to record every step of the training process. This is stated in the logdir variable and then called with the tf.keras.callbacks.TensorBoard.

For this model we define certain parameters that affect the information and performance of the model. Epochs is how many times the model is trained, having a lot of epochs does not necessarily mean it will be better. How the data is validated is also added to the code, and finally calls the logdir that was created previously.

5.1.6. Evaluation. The last step of creating a model is the evaluation of the performance. First the Loss of the model and later the Accuracy. Matplotlib is used for this task because it allow us to compare information easily and graphically.

Then the model is tested by loading the trained model and using images that have not been added into the training or testing batches.

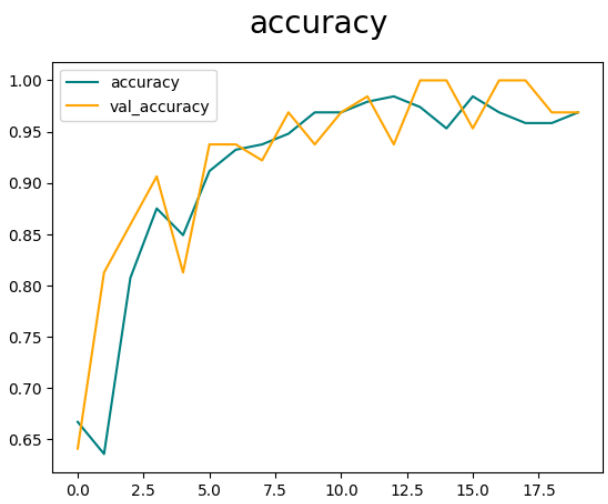

Figure 2. Accuracy graph from the supervised model.

5.2. Unsupervised method

Unsupervised machine learning utilizes algorithms that sort through unlabeled datasets and analyze them. They do not require human intervention and instead work on their own.

As mentioned, for this method we utilized Isolation Forest.

5.2.1. Data collection. The data we are utilizing comes from robotic arm kinematics. We extract the information from the dataset and continue towards the data processing step. The contents of the dataset are going to be used in a Google Colab file.

5.2.2. Node creation. To utilize the Isolation Forest algorithm, we need to create a node class. This class is used to create the nodes of the trees in the forest. Each node represents a decision point where the data is split.

```
def path_length(x, node, current_depth):
    if node is None:
        return current_depth

    if x[node.split_feature] < node.split_value:
        return path_length(x, node.left, current_depth + 1)
    else:
        return path_length(x, node.right, current_depth + 1)

class Node:
    # The __init__ method is the constructor for the Node class.
    # It initializes a new instance of the Node with the given parameters.
    def __init__(self, left, right, split_feature, split_value):
        # self.left: This attribute stores the left child of the node.
        # The left child is a subtree that contains data points less than the split_value.
        self.left = left

        # self.right: This attribute stores the right child of the node.
        # The right child is a subtree that contains data points greater than or equal to the split_value.
        self.right = right

        # self.split_feature: This attribute stores the index of the feature based on which the split is made.
        # In the context of Isolation Forest, it's a randomly selected feature.
        self.split_feature = split_feature

        # self.split_value: This attribute stores the value used to split the data at this node.
        # Data points with a value less than split_value for the split_feature go to the left child,
        # and others go to the right child.
        self.split_value = split_value
```

Figure 3. Node class.

```
class IsolationTree:
    def fit(self, X, depth=0, max_depth=10):
        # If the current depth equals or exceeds the maximum depth, or if the dataset X has 1 or fewer points,
        # the recursion stops and returns None, indicating that this node is a leaf.
        if depth >= max_depth or len(X) <= 1:
            return None

        # Randomly select a feature (column) from the dataset X for splitting.
        split_feature = random.randint(0, X.shape[1] - 1)

        # Choose a random split value between the minimum and maximum values of the selected feature.
        split_value = np.random.uniform(X[:, split_feature].min(), X[:, split_feature].max())

        # Create a boolean array where True indicates that the data point's feature value is less than the split value.
        left_indices = X[:, split_feature] < split_value
        # The opposite of the left_indices to get the right_indices.
        right_indices = ~left_indices

        # Recursively create the left subtree using the subset of X that satisfies the left_indices.
        left = self.fit(X[left_indices], depth + 1, max_depth)
        # Recursively create the right subtree using the subset of X that satisfies the right_indices.
        right = self.fit(X[right_indices], depth + 1, max_depth)

        # Return a new Node with the left and right subtrees, the split feature, and the split value.
        return Node(left, right, split_feature, split_value)
```

Figure 4. Isolation tree fit.

In this implementation, the fit method recursively builds an isolation tree. At each node in the tree, the dataset is split into two subsets based on a randomly selected feature and a randomly determined split value. This process is repeated recursively for each subset, creating a binary tree structure. The recursion stops when the specified maximum depth is reached or when there is only one data point left in the subset, which is when the method returns None, indicating a leaf node. The essence of this approach lies in its random partitioning of the data, which is particularly effective for anomaly detection since anomalies are ”isolated” with fewer splits compared to normal points. The system will go through the data, detect anomalies that it encounters and at the end, display a graphic with all of the obtained data points arranged in a 3D space.

6. Semi-Supervised

6.1. Catkin and TurtleBot3

Catkin is ROS's official build system succeeding rosbuild. It simplifies building interconnected ROS packages.

We build a special package for our self-navigating robot using catkin_make. A custom built package, named ”auto_nav,” helps different parts of our robot talk to each other. It makes sure the sensors, controls, and decision-making parts can share information smoothly.

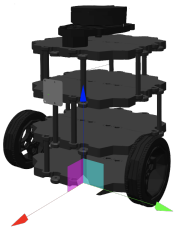

Figure 5. TurtleBot Burger robot

TurtleBot3 is a small, affordable, and programmable ROS-based robot used in education, research, and product prototyping. It aims to maintain quality and features while reducing size and cost. Its design allows customization using various mechanical and optional parts like sensors and computers.

The core technology of TurtleBot3 includes SLAM (simultaneous localization and mapping), navigation, and manipulation. It's structure can allow it to create maps, navigate rooms, and even manipulate objects given the appropiate modifications and programming.

We looked at how the TurtleBot3 moves around and used that as a starting point for creating our own way of figuring out where it is and making maps. This saved us a lot of time because we didn't have to build a whole new robot or figure out how it moves in simulations. While we made changes to the mapping part, we borrowed the way it moves to help our own system.

6.2. SLAM Gmapping

GMapping tackles the Simultaneous Localization and Mapping (SLAM) challenge by using a Particle Filter (PF), it figures out both the map and where the robot is on that map. They create a group that tries to match the real probability distribution, based on Importance Sampling. This deals with the likelihood of the map and the robot's position considering control inputs (like motor encoder counts) and sensor readings (like LiDAR). There's also a motion model and a sensor model playing a role in this probability calculation.

The robot initializes a set of particles (hypothesized robot poses) randomly across the map. Each particle represents a potential position and orientation of the robot. We estimate its next position using motion models (e.g., odometry information from wheel encoders). Each particle's pose is updated based on these motion models, simulating the expected movement of the robot. When the robot senses its surroundings (e.g., through sensors like lidar, cameras, etc.), it uses these measurements to correct the estimated positions of the particles. This involves a comparison between the expected sensor readings from each particle's pose and the actual sensor readings obtained by the robot, particles that better match the sensor measurements are given higher weights, representing their likelihood of being closer to the true robot pose. Particles are replicated based on their weights. Particles with higher weights are duplicated more often, while those with lower weights might be discarded.

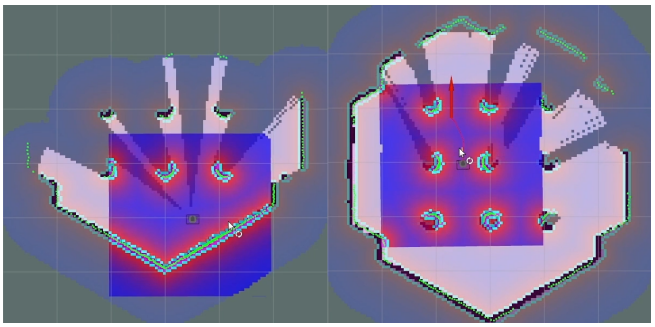

Figure 6. TurtleBot Burger robot mapping out the room as it moves

Over time, as the robot gathers more sensor data, the particles converge toward the actual robot pose and the

map of the environment. During this process, the algorithm simultaneously builds a map of the environment based on the collected sensor data and the estimated robot poses.

The strength of Particle Filter SLAM lies in its ability to handle non-linearities and uncertainties in both robot motion and sensor measurements. However, it requires computational resources because of the need to maintain and update multiple particles representing potential robot poses.

7. Results

7.1. Supervised

The loss graph shows both the training loss and the validation loss. Both losses decrease over time, indicating the model is learning and improving. There is no sign of overfitting. The validation loss follows the training loss closely. The model shows signs of convergence, as the loss values stabilize towards the later epochs. The

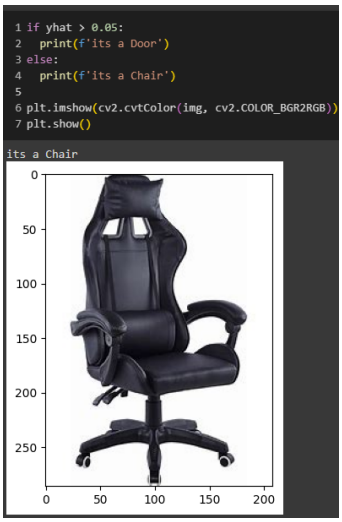

Figure 7. Prediction from the supervised model.

accuracy graph displays the training accuracy and validation accuracy. Both accuracies increase sharply at the beginning and then flats, which is typical as the model starts to reach the performance limit. The validation accuracy closely tracks the training accuracy, suggesting that the model is not overfitting. The accuracy appear to fluctuate slightly in the later epochs, which is common as the model fine-tunes its parameters. The model has an appropriate accuracy and can differentiate between chairs and doors.

7.2. Unsupervised

As we can appreciate, there were not many anomalies in the tested data, some of the anomalies are also obscured by the normal data points. This is also because we set a high parameter in the depth of the tree and other values. This type of models can not be tested for accuracy because they are designed to agroup data and create clusters. Our created model finds anomalies in the robots behaviour that later can be studied to find what these are, outliers, errors in the sensor readings, signs of maintenance, and many more possibilities.

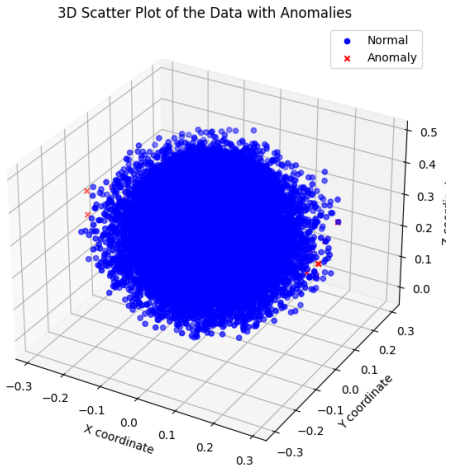

Figure 8. Red marks indicate anomalies.

7.3. Semi-Supervised

The robot mapped its surroundings without significant delay. Its movement was smooth, successfully navigating to manually specified points on the map. However, obstacle avoidance through SLAM requires more thorough exploration. Due to time constraints, comprehensive testing in this area wasn't possible. Nevertheless, the robot's ability to recognize the environment remains functional.

8. Conclusion

8.1. Supervised

The desired model that we stated in the beginning was a neural network that could differentiate several house objects like fridges, doors, chairs, and many more. However, this resulted in a very complex training and we took the decision to only classify two objects. We faced a lot of problems for the training as the initial images were too heavy and then switched to lighter ones.

8.2. Unsupervised

Choosing the model was difficult to determine due to unsupervised machine learning needing much more time to train, therefore we required a simpler model that could be extrapolated to our cleaning robot. We faced several challenges to implement from scratch because the tree node required knowledge of Nodes and Classes of python, which we could not understand at all but the necessary for creating the model.

8.3. Semi-Supervised

Catkin provides a platform for crafting personalized ROS packages, while Turtlebot3 lended us a look inside movement algorithms and message exchanges for robot control. SLAM ensables a robot to identify and chart its surroundings for autonomous navigation.

However, due to time constraints, the autonomous movement aspect wasn't fully done. Instead, the focus was on mapping areas using gmapping SLAM. Developing completely autonomous movement would have required further investigation and a more extensive coding effort.

References

[1] ROS Wiki Contributors (2023, January). "Catkin." ROS Wiki. Available: http://wiki.ros.org/catkin. [Accessed: December 5, 2023].

[2] ROBOTIS (2023, January). "SLAM Simulation." Turtlebot3 e-Manual. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/slam_simulation/. [Accessed: December 5, 2023].

[3] S. Thrun, "Particle Filters in Robotics," in Proceedings of Uncertainty in AI (UAI), 2002. [Online]. Available: http://robots.stanford.edu/papers/thrun.pf-in-robotics-uai02.pdf. [Accessed: December 5, 2023].

[4] N. Renotte, "Build a deep CNN image classifier with ANY images," 25-Apr-2022. [Online]. Available: https://www.youtube.com/watch?v=jztwpsIzEGc. [Accessed: 05-Dec-2023].

[5] N. Renotte, ImageClassification: Jupyter notebook showing how to build an image classifier with Python and Tensorflow. Available: https://github.com/nicknochnack/ImageClassification. [Accessed: 05-Dec-2023].

[6] Akshara, "Anomaly detection using Isolation Forest - A Complete Guide," Analytics Vidhya, 26-Jul-2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/07/anomaly-detection-using-isolation-forest-a-complete-guide/. [Accessed: 06-Dec-2023].

[7] Researchgate.net. [Online]. Available: https://www.researchgate.net/publication/224384174_Isolation_Forest. [Accessed: 06-Dec-2023].