ÇANKAYA UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

CENG 407

LITERATURE REVIEW

A Review of the Computer Science Literature Relating to Autonomous Discovery Vehicles

Members

Turgut Utku ALTINKAYA

Burak ATEŞ

Yunus Emre DİNÇEL

Bayram Alper KILIÇ

İlteriş SAMUR

Table of Contents

1. Introduction

Autonomous robotic systems that can map and navigate unknown areas can be advantageous in various fields such as search and rescue, agricultural operations, and industrial automation. The aim of this project is to develop an autonomous vehicle that can perform 3D mapping and autonomous movement using Raspberry Pi, PiCamera, and LIDAR sensor. The two main components used in the development of this project are ROS (Robot Operating System) and SLAM (Simultaneous Localization and Mapping) and these are used for mapping purposes as well as enabling the localization of the vehicle. The role of Raspberry Pi in this case is to act as a publisher that sends the information obtained from the sensors and sends it to a remote computer. PiCamera and LIDAR sensors provide basic visual information and depth information about the environment for mapping and identification of obstacles.

While the vehicle operates autonomously, its environment is mapped to a 3D model using an application named RTAB-Map. The generated map will be visible in real-time using RViz. After the generation of the 3D map, we will classify the objects that are found in that environment. This not only makes the environment clearer for the robot, but also provides a clear and active strategy in which autonomous systems can be implemented. The results of the project will provide evidence of the ability of Raspberry Pi, ROS, and SLAM to map an unknown environment in 3D, which can then be used in rescue operations, defense industry, etc.

2. Autonomous Movement

Autonomous vehicles rely on their perception systems to acquire information about their immediate surroundings. It is necessary to detect the presence of other vehicles, pedestrians and other relevant entities. Safety concerns and the need for accurate estimations have led to the introduction of Light Detection and Ranging (LIDAR) systems in complement to camera or radar based perception systems [1].

2.1 What is LIDAR?

The LIDAR (Light detection and ranging) utilizes the laser phenomenon to detect objects. The sensor emits laser pulses (aka light beams) to the surrounding environment, the reflected lights are captured by the sensor. After that, the light speed and time elapsed is used to calculate distance from the object by the LIDAR sensor [2].

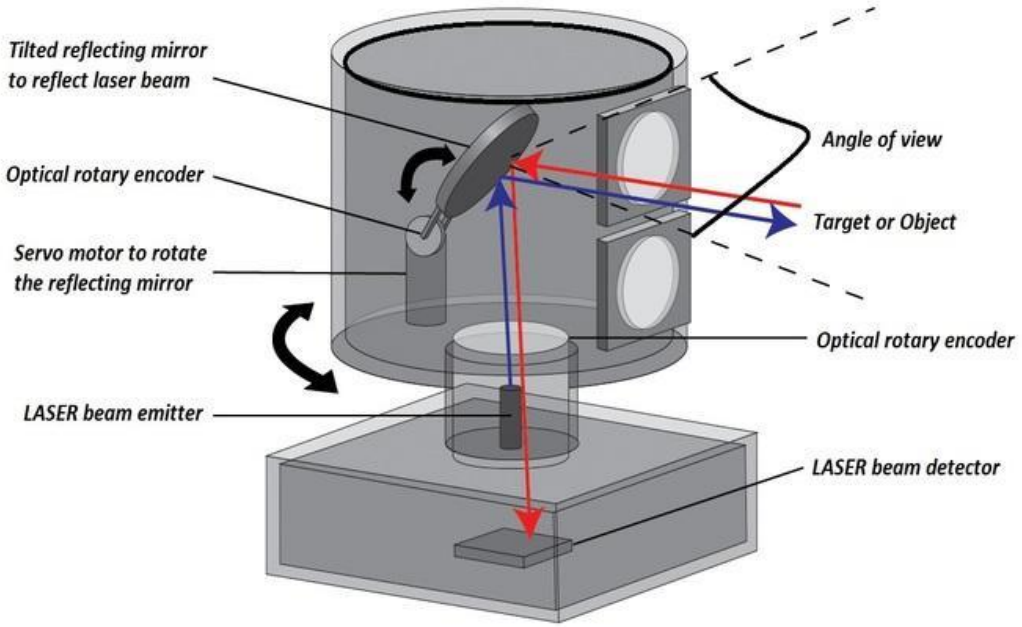

Figure 1: LIDAR Sensor Structure

2.2 Object Detection

The vehicle uses a LIDAR sensor to detect and map nearby obstacles by the LIDAR sensor attached to itself. This allows it to measure distances, creating a real-time 3D map. When an obstacle is detected, the vehicle's algorithms adjust its route, enabling it to navigate safely and avoid collisions, even in challenging environments.

2.3 Movement & Object Avoidance

The vehicle can move and discover autonomously, but moving and detecting objects is not sufficient for an autonomous vehicle. It also needs to avoid the obstacles which are detected by the LIDAR. There are some algorithms which are used to avoid obstacles while reaching the endpoint.

2.3.1 Bug Algorithm

The Bug algorithm is a simple obstacle avoidance technique for robots navigating toward a target. When the robot encounters an obstacle, it follows the obstacle's boundary until it finds a point closer to the goal. It then resumes moving directly toward the target.

There are different variations, like Bug1 and Bug2:

- **Bug1**: The robot circles the obstacle entirely before deciding on the optimal exit point.

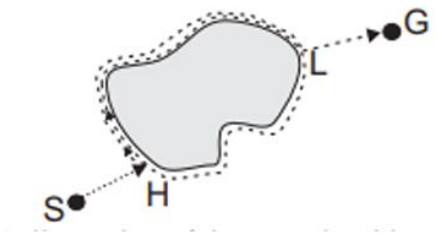

Figure 2: Bug1 Algorithm

- **Bug2**: The robot follows the obstacle boundary only until it finds a point on the line between its start and the goal.

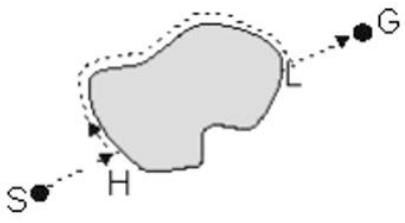

Figure 3: Bug2 Algorithm

2.3.2 Probabilistic Road map (PRM) Algorithm

PRM is a simplified sample-based algorithm which works by constructing a probabilistic road map comprising networks of connected nodes in a given map based on free and occupied spaces to find an obstacle free path from start to end point. Once the roadmap is constructed, a path connecting source to destination is extracted using graph searching techniques [3].

2.3.3 Artificial Potential Fields

The artificial potential field method guides a robot by generating virtual forces that influence its movement. An attractive force pulls the robot toward its goal, while repulsive forces push it away from obstacles to prevent collisions. The robot's direction and speed are determined by the combined effect of these forces, allowing it to navigate smoothly. However, this method can struggle with "local minima," where attractive and repulsive forces balance out, potentially trapping the robot without a clear path to the goal [3].

2.3.4 Obstacle Avoiding Algorithms Comparison Table

Algorithm	**Advantage**	**Disadvantage**
Bug Algorithm	Simple and easy to implement. Effective in small, static environments.	Inefficient in complex or dynamic environments. Can get stuck in local minima.
PRM Algorithm	Fast, continuous path adjustment. Smooth, real-time navigation.	Can get stuck in local minima. Struggles with multiple obstacles creating conflicts.
APF	Works well for large, complex environments. Effective for global path planning.	Computationally expensive. Not ideal for real-time navigation in dynamic environments.

3. Path Planning

Path planning is a crucial component for autonomous vehicles to navigate efficiently and safely within diverse and complex environments. Autonomous vehicles rely on the path planning algorithms to create routes that avoid obstacles by minimizing travel cost like time, distance, and energy. The path planning algorithms are designed to meet the these requirements by generating optimal paths to the vehicle's environment, capabilities, and constraints. According to the environment, path planning approaches can be categorized into global and local methods. Global path planning algorithms are used when the environment is known. The algorithms compute a complete route from start to finish. The global path planning algorithms are suitable for static environments where the obstacles and boundaries are predictable. Local path planning algorithms, however, are essential in dynamic or unknown environments, where real-time adjustments must be made based on the sensor data as new obstacles or changes in the environment are detected.

An efficient path planning algorithm depends on factors like, computational resources, complexity of the environment and the need for real-time responsiveness. For static environments, classical approaches like Dijkstra's and A* are suitable since the environment is mapped well and can produce optimal paths with relatively low computation. Dynamic or unknown environment, on the other hand, more adaptive techniques like D*, RRT (Rapidly-Exploring Random Trees), Genetic Algorithm, ACO (Ant Colony Optimization) and Firefly algorithms are used [4].

3.1. Dijkstra's Algorithm

Dijkstra's algorithm is a classic graph search based algorithm to find the shortest path between node in a graph. The algorithm dives node into two sets, "visited" and "unvisited". The starting node assigned distance of zero, while others are set to infinity. Algorithm updates of each neighboring node if a shorter path is found. After updating neighbors, the current node moves to the "visited" set, and the process repeats until all nodes are visited [5]. Dijkstra's algorithm well perform on the static environments where the map is known in advance. However, the algorithm can be computationally expansive for large graphs in order to travel various nodes

.

3.2. A* Algorithm

A* is an informed based algorithm used to finding the shortest path. It operates similarly to Dijkstra's algorithm, but A* extends the algorithm by adding heuristics cost. A* algorithm evaluates node by calculating a function $f(n) = g(n) + h(n)$, where:

- $g(n)$ is the cost of path from start node, to the current node.
- $h(n)$ is the heuristic estimate of the cost of the current node to the goal node.

Despite Dijkstra's algorithm, A* algorithm prioritize nodes that are promising, guiding the search towards the goal while ensuring the path remains optimal and saving a significant amount of computation time [6].

3.3. D* Algorithm

The D* (Dynamic A*) algorithm is a path planning technique designed for dynamic or unknown environments where obstacles can appear or change as the autonomous vehicle navigates [7]. Unlike, traditional algorithms like Dijkstra and A*, which are re-computes paths from scratch, D* adapts by updating only the parts of the path that are affected by changes. This localized update significantly enhances computational efficiency.

The D* algorithm have operations which are "NEW", "OPEN", "CLOSED", "RAISE", and "LOWER" [8].

- NEW, meaning it has never been placed on the OPEN list.
- OPEN, meaning it is currently on the OPEN list.
- CLOSED, meaning it is no longer the OPEN list.
- RAISE, indicating its cost is higher than the last time it was on the OPEN list
- LOWER, indicating its cost is lower than the last time it was on the OPEN list

D* algorithm iteratively selects and evaluates nodes from the "OPEN" list, starting from the goal node and working backwards towards the start. Each expanded node has a back pointer to the next node towards the target, and it knows the exact cost to reach the goal. When an obstruction is detected along the path, affected nodes are added to "OPEN" list and marked as "RAISED". Before increasing, the cost of a "RAISED" node, the algorithm checks its neighbors to see if the cost can be reduced. If not, "RAISED" state is propagated to its descendants, which are nodes with back pointers to it. These nodes are then evaluated, and the "RAISE" state continues to spread. If a "RAISE" node's cost can be reduced, its back pointer is updated, and the "LOWER" state is passed to its neighbors [9].

3.4. Rapidly-Exploring Random Tree

The Rapidly-Exploring Random Tree (RRT) algorithm is a sampling based path planning algorithm method designed to efficiently search high-dimensional spaces, particularly for system with complex dynamics and constraints. RRT is used in autonomous vehicles in part of path planning due to its ability to explore large configuration spaces rapidly.

RRT operates by incrementally constructing a tree rooted at the initial configuration and expanding toward random samples within the search space. At each iteration, a new point is randomly selected, and the algorithm identifies the nearest node in the current tree. The tree then extends from this nearest node toward the sample point, constrained by the system's dynamics. This expansion is toward unexplored regions, allowing RRT to cover the configuration space effectively without extensive prior knowledge [4].

The advantages of the RRT algorithm lies in its speed and ease of implementation, It is relatively fast compared to other path planning algorithms. Its main cost is the search for closest neighbor, which becomes more computationally intensive as the number of generated vertices increases [10].

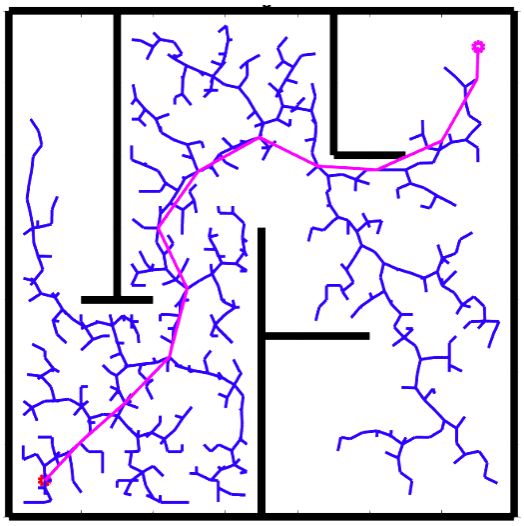

Figure 4: Rapidly-Exploring Random Tree

3.5. Genetic Algorithm

The Genetic Algorithm (GA) are adaptive heuristic search methods within evolutionary algorithms inspired by natural selection and genetics. These are intelligent exploitation of random searches, using historical data to guide the search toward higher performing regions in the solution space [11].

The GA process starts with a population of random paths, evaluated by a fitness function for criteria like path length, obstacle avoidance, or smoothness. Through selection, crossover and mutation, GA iteratively refines the population. Selection chooses high-fitness individuals as “parents” for the next generation, crossover combines parent traits to enhance diversity, and mutation introduces small random change to explore new areas in the search space.

The fitness function plays a critical role in directing the algorithm towards to optimal solutions. For path planning, it typically incorporates constraints like distance, obstacle avoidance, and path smoothness, with each factor weighted according to priority. Over successive generations, GA converges on a set of solutions that balance these factors, progressively refining the path quality.

3.6. Ant Colony Optimization

The Ant Colony Optimization (ACO) is a natural-inspired optimization algorithm. It is a suitable option for path planning, especially in complex, changing environments. Modeled on the natural behavior of ants locating the shortest path to food [12]. ACO operates by simulating “artificial ants” that traverse a graph from starting point to a target, searching for the shortest path while avoiding obstacles. As ants move, they deposit virtual “pheromones” on the edges of the graph, which act as probabilistic cues to guide subsequent ants. Over time, paths with higher pheromone concentrations attract more ants, reinforcing the desirability of shorter, more efficient routes. Conversely, pheromones on less-traveled paths cleared, allowing the algorithm to avoid local optima by reducing the likelihood of ants choosing suboptimal paths [4].

3.7. Firefly Algorithm

The Firefly Algorithm (FA) is metaheuristic optimization technique inspired by the communication and social behavior of fireflies. It has been applied efficiently to path planning for autonomous vehicle systems. The core principle of FA is that fireflies are attracted to brighter ones, moving towards those with higher intensity in order to optimize their path based on specific objectives like distance, safety, and smoothness [13].

The Firefly Algorithm (FA) helps autonomous vehicle systems navigate by balancing exploration with optimization. Each firefly (or solution) is attracted to better options, adapting paths dynamically for complex and unknown environments. FA's mix of randomness and directed movement enables effective handling of obstacles, while modifications like decreasing step size enhance convergence speed and efficiency, making FA suitable for real time autonomous vehicle [4].

4. 3D Mapping

3D mapping is a process in robotics that involves creating a three-dimensional representation of an environment. Unlike traditional 2D mapping, which only captures flat layouts, 3D mapping provides a full representation of the environment. 3D mapping is crucial in applications such as autonomous navigation, object detection, and obstacle avoidance, which require accuracy and a deep understanding of depth for effective robot operation. Robotic systems usually utilize LIDAR sensors and cameras to create 3D maps of environments. These sensors provide distance, depth, and visual data to robotic systems for constructing maps of the environment. In this context, ROS (Robot Operating System) is crucial. ROS provides frameworks and tools for integrating sensors and processing 3D mapping data. The most commonly used technique for this purpose is SLAM (Simultaneous Localization and Mapping). SLAM enables robots to build a 3D map of their environment while simultaneously tracking their own location within it. By utilizing SLAM algorithms, robots can navigate unknown spaces and adapt to changing environments. This combination of ROS and SLAM, along with data from LIDAR sensors and cameras, allows robotic systems to construct accurate and dynamic 3D maps essential for autonomous tasks.

4.1. What is ROS?

ROS (Robot Operating System) is an open-source robotics middleware suite. ROS is not an operating system. ROS is a set of software frameworks for robot software development. It provides tools for managing different hardwares and computers working together with hardware abstraction, low-level device control, message passing and hardware management.

ROS processes are represented as nodes in a graph structure connected by edges called topics. Nodes can represent various components in a robotic system such as sensors, services, actuators, and decision-making units. ROS nodes can pass messages to other ones through a publish-subscribe model. Nodes can publish messages to specific topics, and other parts of the system can subscribe to these topics to receive the data in real-time [14].

4.1.1. The Role of ROS in 3D Mapping

The ROS has an important role in 3D mapping by providing the frameworks for managing the SLAM algorithm. ROS facilitates the deployment and operation of SLAM algorithms through its middleware capabilities. Specifically, ROS enables the following key functions in 3D mapping:

Sensor Integration and Data Processing: ROS can handle and synchronize multiple data sources such as sensors and cameras. ROS will configure, process and forward the data to nodes that handle the SLAM algorithm.

SLAM Algorithms Frameworks: ROS provides the structured environment to implement various SLAM algorithms, such as GMapping, Hector SLAM, Karto SLAM, and RTAB-Map. These frameworks use sensor and camera data for building 3D maps with SLAM algorithms.

Simulation Environment: The ROS includes simulation frameworks like Gazebo. This simulation environment enables the testing of SLAM algorithms with virtual settings. This capability of ROS enables the replicate real-world conditions in simulation before deploying the hardware.

Loop Closure: Loop closure basically helps a robot understand that an object has already been visited, and therefore the robot will update its location and the map accordingly. The ROS mapping frameworks can detect loop closures [15].

4.2. What is SLAM?

Simultaneous Localization and Mapping (SLAM) was first coined as a term in a scientific paper presented at the 1995 International Robotics Research Symposium. The Simultaneous Localization and Mapping problem asks if it is possible for a robot to be placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map.

The basis of the SLAM is Bayesian Probability, as it provides a way to update beliefs with respect to new information. In SLAM, the robot constantly refines its understanding of its position and the map as it gathers more observations. Bayesian probability helps by combining prior beliefs (the robot's current estimate of where it and the landmarks are) with new evidence (sensor readings or movement data) to produce an updated belief. Bayes Theorem is used for calculate the posterior probability which is:

$$P(xk, m \mid Z0{:}\,k, U0{:}\,k, x0) = \frac{P(Z0{:}\,k \mid xk, m) \cdot P(xk, m \mid U0{:}\,k, x0}{P(Z0{:}\,k \mid U0{:}\,k, x0)}$$

where:

$\boldsymbol{P(xk, m \mid Z0{:}\,k, U0{:}\,k, x0)}$ is the posterior: An updated, more refined belief of the robot's position and the map after incorporating new observations.

$P(Z0{:}\,k \mid xk, m)$ is the likelihood: The probability of observing the new sensor data if the current estimate were correct.

$P(xk, m \mid U0{:}\,k, x0)$ is the prior: The initial probability of the robot's state and map before incorporating the latest observations.

P(Z0:k|U0:k,x0) is the evidence: The normalization factor that adjusts the overall probability distribution, ensuring that the updated beliefs about the robot's position and map are valid and sum to 1.

As well as The Bayes Probability, The Kalman Filter also helps SLAM to predict and correct the robot's estimates of its location and the map. The Kalman Filter works recursively therefore, The robot will update its estimates step by step as new data comes in, which is essential for real-time navigation. Kalman Filter will predict the new state of the robot (location and map) with respect to previous state and movement commands. When a robot observes a new landmark, the Kalman filter compares this observation with its prediction. If there's a difference between the prediction and the observation, the Kalman filter adjusts the estimate. Also the Kalman filter also maintains a covariance matrix that represents the uncertainty in the robot's estimate of both its location and the landmarks. This matrix grows or shrinks based on how much new information comes in [16].

4.3. Analyses of ROS based SLAM Algorithms

4.3.1. GMapping

GMapping is a 2D SLAM algorithm that utilizes Rao-Blackwellized Particle Filter (RBPF) to build maps with respect to LIDAR data and odometry input. The Rao-Blackwellized Particle Filter uses adaptive resampling technique. This helps to decrease the particle-depletion problem and computational complexity. The GMapping algorithm is designed in such a way that it integrates the current sensor data with the odometry motion model. This helps the particle filter's prediction step of uncertainty about the robot's location to decrease. GMapping is sensitive to odometry errors, which can affect map quality. It's not useful for challenging environments.

4.3.2. Hector SLAM

Hector SLAM is a SLAM algorithm which integrates laser scan matching feature with the 3D navigation method by the help of the inertial system which employs the EKF (Extended Kalman Filter). The algorithm computes a 3D state estimation of the robot's position and orientation using the Extended Kalman Filter. Hector SLAM can compute the 6 degrees of freedom of the AGVs position and orientation during motion and parallelly the high update rate laser-based 2D map. It is a SLAM algorithm which does not use the odometry data. Thus, the Hector SLAM has an advantage when used in environments which exhibit the pitch and roll characteristics.

4.3.3. Karto SLAM

Karto SLAM is a graph-based algorithm that incrementally builds maps by optimizing the robot's current pose with respect to previously mapped areas. The major disadvantage with the Kato SLAM is that the complexity of computation is proportional to the number of landmarks. As the number of landmarks increases, the computational complexity also increases.

4.3.4 RTAB-Map

RTAB-Map is a SLAM algorithm which can be implemented with any type of sensor that provides depth information or three-dimensional information. The RTAB-Map is based on the RGB-D Graph SLAM. It can be implemented with 3D lasers, Stereo vision and even with RGB-Depth cameras. This method has a global loop closing detection technique in-built to compute the vehicle pose and an effective memory managing system. The map is saved as a structured graph with the robot's position and orientation and the captured image at that pose as nodes. The Map also contains the odometry data or the loop closing matrices as its edges. The major disadvantage is that the complexity of computation increases with the increase in the size of the map. The RTAB-Map SLAM can also be implemented without the need for odometry data.

SLAM	**Sensors**	**Classification**	**Based On**	**Odometry Needed**	**Loop Closure**
GMapping	2D Lidar	Particle filter	FastSLAM	YES	YES
Hector SLAM	2D Lidar	Kalman filter	EKF	NO	YES
Karto SLAM	2D Lidar	Optimization based SLAM	Graph SLAM	YES	YES
RTAB-Map	3D Lidar, camera	Optimization based SLAM	RGB-D Graph SLAM	NO	YES

In a study analyzing and evaluating various algorithms for autonomous ground vehicles, their performance was examined in both real-time and simulated environments. The research focused on aspects such as mapping accuracy, quality, and adaptability to complex settings. Among the algorithms studied, GMapping was identified as the most effective for generating maps in real-time and simulation environments with respect to SSIM. The SSIM is a metric that will compare the two maps which are of the same size, i.e., length and breadth. The SSIM is a metric which calculates the mean value, the variance and the covariance of the intensities of the cells obtained from the image provided [17].

SLAM	**SSIM (Simulation vs Ground truth)**	**SSIM (Real-Time vs Ground truth)**
GMapping	0.84	0.83
Hector SLAM	0.81	0.78
Karto SLAM	0.75	0.77
RTAB-Map	0.81	0.76

5. Object Classification

Object Classification determines which specific objects are in an image or video. It labels these objects. [18] This technology, which is frequently faced in autonomous vehicles, plays an important role in decision-making to prevent dangerous situations by interpreting the data collected by the vehicle's sensors and cameras and labeling the objects it may encounter while driving. Object localization specifically tracks where objects are located in an image or video. This determines the location of any object within a piece of visual content. For example, for driving an autonomous car, pedestrians, other vehicles around, and traffic lights are of great importance in decision-making during autonomous movement.

With Autonomous Discovery Vehicles, unlike self-driving cars, object detection, and object classification can be used for different purposes. In common with self-driving vehicles, it is aimed at the discovery vehicle to detect dangerous objects and perform mapping and discovery tasks safely by using LIDAR and camera data. In addition, object detection is of great importance in the object-oriented semantic mapping approach on the 3D Map created using SLAM or other algorithms.

5.1 Object-Oriented Semantic Mapping

By using object detection methods to integrate semantic information on a 3D map created with the SLAM algorithm, the Object-Oriented Semantic Mapping approach, which provides a more advanced 3D map, makes it possible for Discovery Vehicles to make more meaningful mapping. This Semantic Mapping approach first geometrically segments incoming frames to create an initial 3D map, then overlays detected objects using bounding boxes, significantly enhancing object boundary definition.

Nakajima and Saito used various technologies and techniques in their approach to Object-Oriented Semantic Mapping [19]. Traditional semantic mapping systems, like those using CNN models, provide detailed scene understanding but often struggle with high computational costs, limiting real-time applicability in autonomous exploration. Nakajima and Saito's work provides an efficient alternative by combining object detection with incremental 3D map segmentation, reducing computational complexity without sacrificing accuracy. Also, this approach enhances real-time object-oriented segmentation and provides valuable insights into creating high-quality 3D maps suitable for real-world navigation and obstacle recognition.

In this approach, incoming RGB-D frames are processed to estimate the camera pose via the Iterative Closest Point (ICP) and RGB alignment methods. This camera pose helps to accurately fuse new depth information into the 3D map, enabling a progressively built, dense representation of the scene. Another advantage of the approach is instead of applying 2D semantic segmentation directly to each frame, which can be slow, they use a geometric segmentation approach based on depth information. This technique groups geometrically similar areas into regions, which are incrementally segmented as new frames are processed.

By performing geometric segmentation on the 3D map, clear object boundaries can be established without needing prior object models. This method first organizes the 3D space into segments based on depth and shape features, creating a foundation for adding semantic labels. Also using the semantic information improves the segmentations. For instance, if two adjacent segments fall within the same bounding box and have similar semantic labels, they are merged. This process includes calculating a confidence score based on the probability that two segments represent the same object, thus refining the map accuracy and ensuring consistent labeling. To complete the object detection, YOLO outputs bounding boxes and class probabilities for objects detected in each frame. These bounding boxes are then used to assign semantic information to segments of the 3D map. After all this process the general view of this approach like in the Figure.

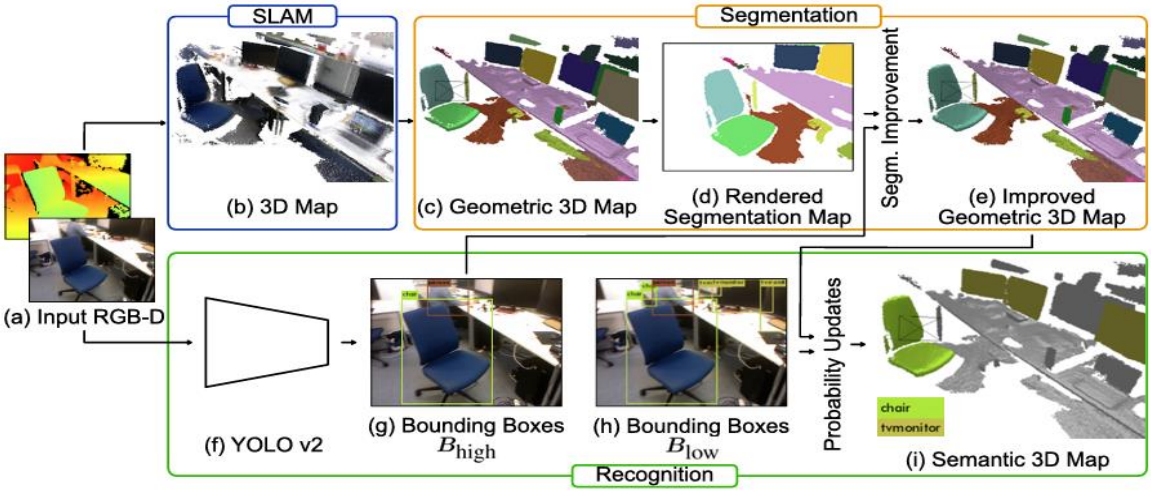

Figure 5: Object Oriented Semantic Mapping

5.2 YOLO (You Only Look Once)

YOLO is one of the most popular model architectures and algorithms for object detection. The YOLO model uses one of the best neural network archetypes to produce high accuracy and overall speed of processing. This speed and accuracy are the main reason for its popularity [20]. YOLO divides an image into a grid and simultaneously predicts bounding boxes (the coordinates of potential objects) and class probabilities (what each object is likely to be) for each grid cell. This “one-shot” approach makes YOLO fast, and ideal for real-time applications. Each grid cell predicts multiple bounding boxes, each with a confidence score. This score represents the likelihood of an object within that box and the accuracy of the box's location. YOLO also assigns class probabilities to each detected bounding box, indicating what object class (e.g., person, car, chair) is most likely in that region. The architecture of YOLO allows the model to learn and develop an understanding of numerous objects more efficiently. Considering the Object-Oriented Semantic Mapping Results, YOLO effects can be easily observed with details in this Figure.

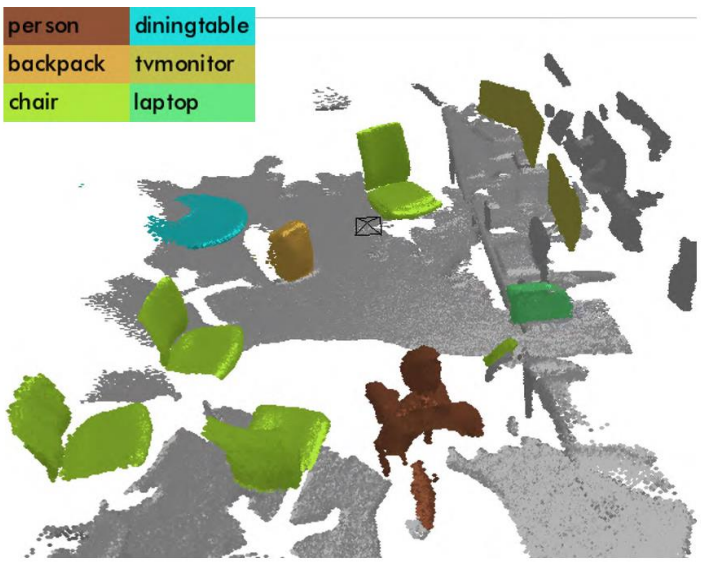

Figure 6: YOLO

6. Related Works

The development of autonomous vehicles has led to profound advances in 3D object detection, map creation, and path finding algorithms. In our project, we aim to produce a vehicle that maps its surroundings in 3D and finds its way autonomously. Before implementing our project in practice, it is very important to read past studies and work on the subject. We can take these studies as examples when developing our project. In this section, related studies will be explained in detail.

6.1 Sensor Fusion for Enhanced 3D Perception

3D Maps are generated using the sensor fusion [21], which integrates LIDAR and camera data. The combination of LIDAR's depth data with the visual details from cameras provides a more comprehensive perception of surroundings. This technique not only addresses the limitations of the individual sensors, but also allows us to detect objects even under varying lighting and environmental conditions. An application named RTAB-MAP [22] (Real-Time Appearance-Based Mapping) incorporates sensor fusion algorithms to create detailed 3D maps for SLAM (Simultaneous Localization and Mapping). This application can be run using ROS [23] (Robot Operating System) or on IOS devices. For example, in a project named HURBA [24], they used RTAB-Map to generate a map for a robot simulation using ROS. The generated map can be displayed using RTAB-Map Viz (rviz) in real-time, as the robot generates the map when finding its way autonomously. This is a similar work that we plan to implement in our project. However, they only implemented the robot digitally, meaning that the robot is not present in the real life. Our project aims to implement it in real life, and also add some additional features like classifying the objects on the generated 3D map.

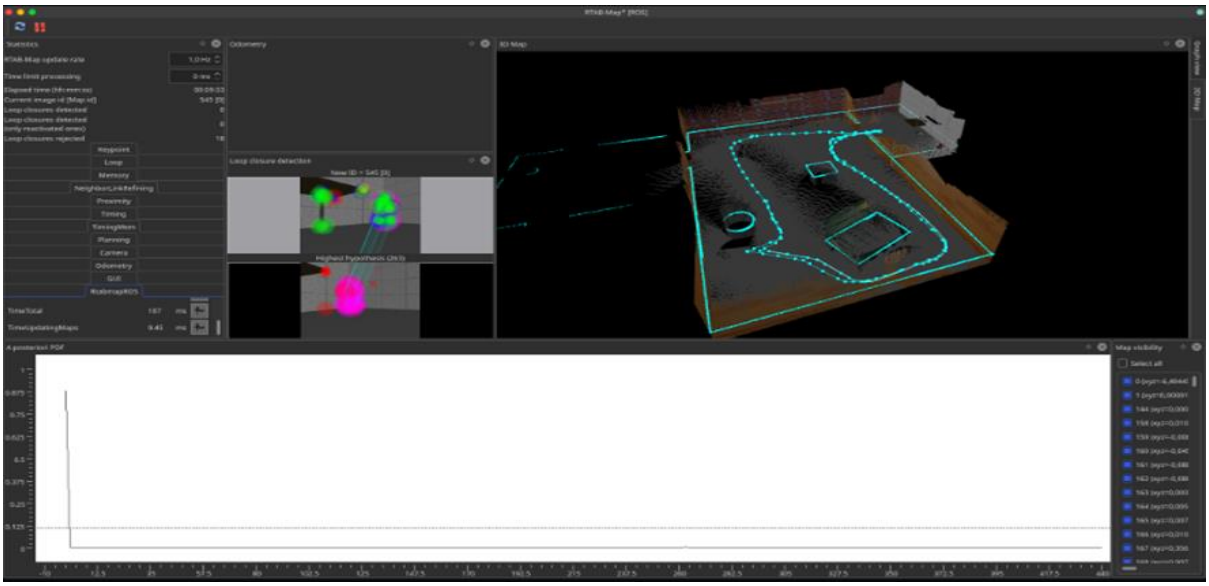

Figure 7: HURBA UI

6.2. 3D-printed robot that uses SLAM for autonomous navigation

Another example project is a 3D-printed robot [25] that uses the SLAM (Simultaneous Localization and Mapping) algorithm to navigate autonomously. This Project also uses ROS and LIDAR to map its surroundings and find its way. However, the project only maps the environment in 2D, which means that objects inside that environment are not detected and classified. In ROS, there can be many nodes where users can write and run scripts, and data can be sent to other ROS applications via publisher nodes. Subscriber nodes can receive messages sent from publisher nodes, so that data from the publisher, which is the robot, can be processed on the remote desktop. In this project, instead of Raspberry Pi, they used Nvidia Jetson Nano, which is more expensive and powerful than Raspberry Pi. Since it is more powerful, they processed the data on that robot and did not send it through publisher nodes. In our project, we aim to send the data to a remote computer so that the data can be processed more efficiently and thus reduce the stress on the robot. The parts of the SLAM configuration with ROS are explained in detail on the project website. For example, the localization and mapping algorithm mainly uses three core frames. They are map, odom and base_link. Map is the frame that represents the real-world environment where the robot should be fixed; odom is the frame that remains fixed relative to the robot's initial position and is used to depict the robot's odometry location estimate. base_link is the frame that represents the robot's location. The map to odom transformation is calculated by the localization node, which mainly depicts the alignment between the sensor data and the incorrect odometry data over time [26]. In this project, they implemented the odom to baselink transformation, representing the difference between the estimated current position and orientation and the real ones.

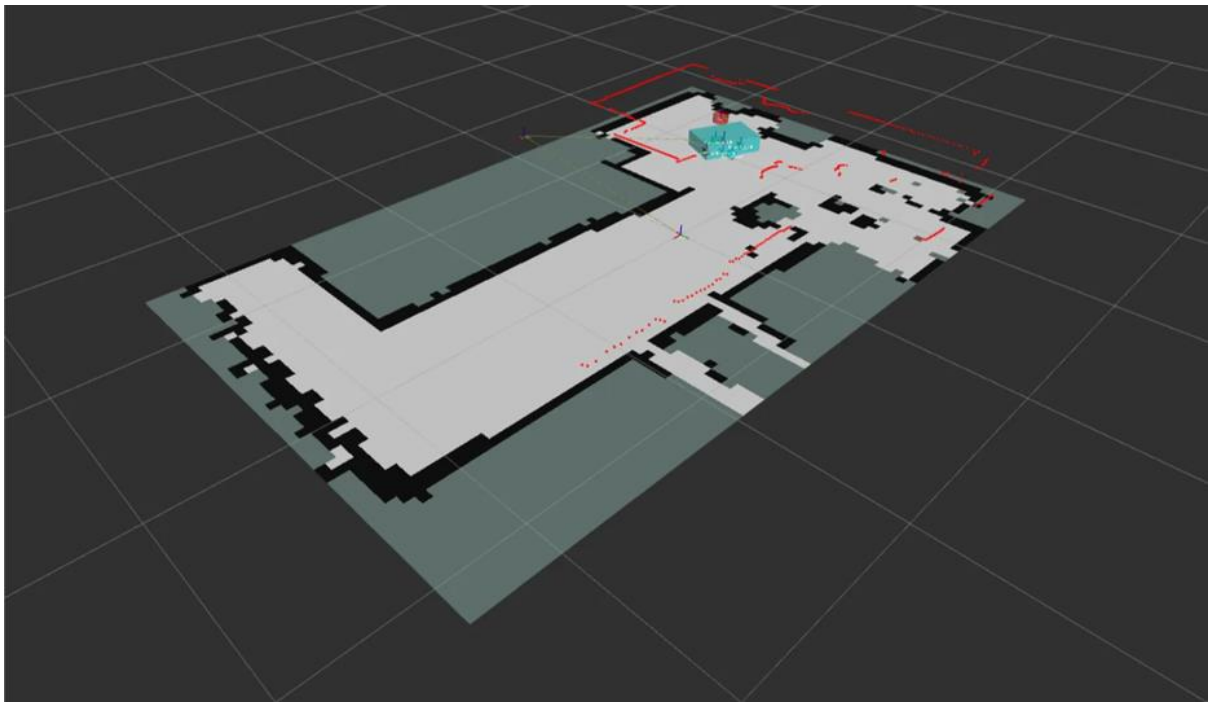

Figure 8: Generated Map

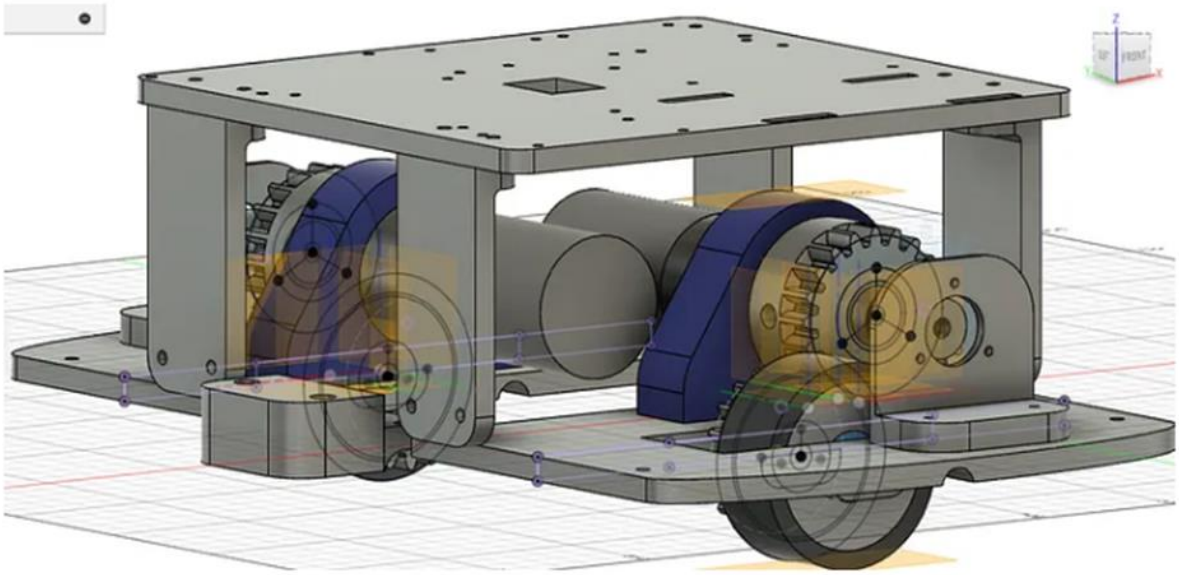

Figure 9: 3D Model Car

This project is really similar to our project. However, since it does not create a 3D Map, we plan to further develop this project, add a 3D mapping using sensor fusion with LIDAR and camera data, and classify the objects detected in that environment while the robot autonomously explores and finds its way.

References

[1] J. I.-G. You Li, "Lidar for Autonomous Driving: The principles, challenges, and trends for automotive lidar and perception systems," 17 April 2020.

[2] S. U. H. Syed, "ResearchGate," ResearchGate, 14 March 2022. [Online]. Available: https://www.researchgate.net/publication/359263639_Lidar_Sensor_in_Autonomous_Vehicles. [Accessed 6 November 2024].

[3] D. S. M. A. P. D. o. T. G. P. ,. D. o. E. G. M. Basavanna M, " An Overview of Path Planning and Obstacle Avoidance Algorithms in Mobile Robots," *International Journal of Engineering Research & Technology (IJERT)*, 18 December 2019.

[4] N. S. C. D. J. E. S. Karur Karthik, "mdpi," 27 May 2021. [Online]. Available: https://www.mdpi.com/2624-8921/3/3/27. [Accessed 5 November 2024].

[5] "Robotic Path Planning," MIT, [Online]. Available: https://fab.cba.mit.edu/classes/865.21/topics/path_planning/robotic.html. [Accessed 6 November 2024].

[6] "A* search algorithm," Ada Computer Science, [Online]. Available: https://adacomputerscience.org/concepts/path_a_star. [Accessed 6 November 2024].

[7] "Path Planning Algorithms for robotic systems," [Online]. Available: https://roboticsbiz.com/path-planning-algorithms-for-robotic-systems/. [Accessed 6 November 2024].

[8] "D*," Wikipedia, [Online]. Available: https://roboticsbiz.com/path-planning-algorithms-for-robotic-systems/. [Accessed 6 November 2024].

[9] D. Biswas, "D*, D* Lite & LPA*," Medium, [Online]. Available: https://dibyendu-biswas.medium.com/d-d-lite-lpa-e7483779a7ca. [Accessed 6 November 2024].

[10] T. Chinenov, "Robotic Path Planning: RRT and RRT*," Medium, 14 February 2019. [Online]. Available: https://theclassytim.medium.com/robotic-path-planning-rrt-and-rrt-212319121378. [Accessed 6 November 2024].

[11] " Genetic Algorithms," GeeksforGeeks, 8 March 2024. [Online]. Available: https://www.geeksforgeeks.org/genetic-algorithms/. [Accessed 6 November 2024].

[12] "Ant colony optimization algorithms," Wikipedia, 22 September 2024. [Online]. Available: https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms. [Accessed 6 November 2024].

[13] A. N. A. K. B. B. M. H. M. N. ,. M. F. A. A. S. A. M. Mohd Nadhir Ab Wahab, "Plos One," 12 August 2024. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0308264. [Accessed 6 November 2024].

[14] "Robot Operating System," Wikipedia, 31 October 2024. [Online]. Available: https://en.wikipedia.org/wiki/Robot_Operating_System. [Accessed 6 November 2024].

[15] "Introduction to Loop Closure Detection in SLAM," Think Autonomous, 18 April 2023. [Online]. Available: https://www.thinkautonomous.ai/blog/loop-closure/. [Accessed 6 November 2024].

[16] T. B. H. Durrant-Whyte, "IEEE Xplore," June 2006. [Online]. Available: https://ieeexplore.ieee.org/document/1638022. [Accessed 6 November 2024].

[17] T. S. H. D. P. P. S. P. P Sankalprajan, "IEEE Xplore," 5 June 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9154101. [Accessed 5 November 2024].

[18] K. Libby, "What is Object Classification and How Can We Use It?," Shutterstock, 14 April 2023. [Online]. Available: https://www.shutterstock.com/blog/object-classification-in-computer-vision. [Accessed 6 November 2024].

[19] H. S. Yoshikatsu Nakajima, "IEEE Xplore," 16 December 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8579143. [Accessed 6 November 2024].

[20] B. K, "Object Detection Algorithms and Libraries," Neptune Ai, 15 April 2024. [Online]. Available: https://neptune.ai/blog/object-detection-algorithms-and-libraries. [Accessed 6 November 2024].

[21] "Yeong, D. J., Velasco-Hernandez, G., Barry, J., & Walsh, J. (2021). Sensor and sensor fusion technology in autonomous vehicles: A review. Sensors, 21(6), 2140.".

[22] "rtabmap," introlab, [Online]. Available: https://github.com/introlab/rtabmap. [Accessed 6 11 2024].

[23] "ROS," [Online]. Available: https://www.ros.org/. [Accessed 6 11 2024].

[24] "Project-2-Mapping," [Online]. Available: https://github.com/hungarianrobot/Project-2-Mapping. [Accessed 6 11 2024].

[25] "3D-Printed-ROS-SLAM-Robot," [Online]. Available: https://github.com/pliam1105/3D-Printed-ROS-SLAM-Robot. [Accessed 06 11 2024].

[26] "Medium," [Online]. Available: https://medium.com/@pliam1105/building-a-3d-printed-robot-that-uses-slam-for-autonomous-navigation-cd83473dac7c. [Accessed 06 11 2024].