Uludağ University Journal of The Faculty of Engineering, Vol. 30, No. 2, 2025 RESEARCH

DOI: 10.17482/uumfd.1592983

HECTOR SLAM BASED NAVIGATION WITH INTEGRATED OBJECT DETECTION

Saran SAPMAZ [*]
Mahmut PEKEDİS [**]

Received: 28.11.2024 ; revised: 27.06.2025 ; kabul: 27.06.2025

Abstract: The main aim of this study is to develop an approach that detects and classifies objects in indoor areas by applying Hector Simultaneous Localization and Mapping (SLAM) and "You Only Look Once" (YOLO) algorithms to an autonomous, custom-made mobile robot. The approach is based on the Robot Operating System (ROS) and computer vision. The mobile robot's motion, path, and communication are controlled by the ROS navigation package. This paper provides a detailed description of the approach implemented to create a new map with objects, as well as information about the hardware and software configuration of the mobile robot. The object identification process and map creation are performed using a low-budget Laser Imaging Detection and Ranging (LIDAR) sensor and non-encoder DC motors. The results show that the proposed technique can detect objects with an accuracy of 97.8% and 94.86% for the x and y cartesian axes, respectively..

Keywords: Autonomous robot, Hector SLAM, ROS, YOLO

Hector SLAM Based Navigation with Integrated Object Detection

Öz: Bu çalışmanın amacı iç mekanlarda Hector Eş Zamanlı Lokalizasyon ve Haritalama (EZLH) ve "Sadece Bir Defa Bak" (SBDB) kullanan özel yapım bir otonom robotun nesneleri algılamasını ve onları sınıflandırmasını sağlayan bir yaklaşım geliştirmektir. Yaklaşım, Robot İşletim Sistemi (RİS) ve bilgisayarla görme teknolojilerine dayanmaktadır. Mobil robotun hareketi, izlediği yolu ve iletişimi, RİS navigasyon paketiyle kontrol edilmektedir. Makale, yeni bir harita oluşturmak için uygulanan algoritmanın ayrıntılı bir açıklamasının yanı sıra, mobil robotun donanım ve yazılım yapılandırmasına ilişkin bilgileri de sunmaktadır. Nesne tanımlama süreci ve harita oluşturma, düşük bütçeli bir Lazer Görüntüleme Algılama ve Mesafe Ölçümü (LGAMÖ) sensörü ile enkodersiz direkt akım (DA) motorlar kullanılarak yapılmaktadır. Sonuçlar, önerilen tekniğin nesneleri sırasıyla x ve y kartezyen eksenlerinde % 97.8 ve % 94.86 doğrulukla tespit edebildiğini göstermektedir.

Anahtar Kelimeler: Otonom robot, Hector Eş Zamanlı Lokalizasyon ve Haritalama (EZLH), Robot İşletim Sistemi (RİS), Sadece Bir Defa Bak (SBDB)

[*] Communication Address: Ege Üni., Fen Bilimleri Enstitüsü, Makine Müh., Ziraat Fakültesi Prof. Dr. Necdet BUDAK Bloğu Kat:3 Bornova - İzmir

[**] Communication Address: Ege Üni., Mühendislik Fakültesi, Makine Müh., Erzene Mahallesi, Ege Ünv. No:172, 35040 Bornova/İzmir

Corresponding Author: Saran Sapmaz (saran.sapmaz1@gmail.com)

1. INTRODUCTION

The number of algorithms deployed for robots has increased over time as robots have become more accessible to everyone. Most of these algorithms are available to the public so that other developers can take advantage of these open-source tools and develop their own algorithms. Simultaneous localization and mapping (SLAM) is one of the open-source tools and is applied to create a map of an unknown environment by simultaneously locating the position of the robot (Blas and Riisgaard, 2005). The ability to create a map in the absence of GPS makes SLAM popular. Over the decades, researchers have used different sensors, cameras, or a combination of both to create maps and localize robots. One of these techniques is Hector SLAM, and this technique spends fewer resources than other algorithms such as gmapping and cartographer because odometry information is not required for map creation and localization in Hector SLAM. Instead, laser streams are used to position the robot within the map using specific configuration files for navigation (Kohlbrecher and Meyer, 2010). These files include properties of the custom robot, expected trajectory, sensor configuration, and ROS topics to execute the navigation (Marder-Eppstein, 2010). Hector SLAM processes in ROS (Robot Operating System) that are used by robotic field researchers to control various robots using standard, ready-to-use algorithms. It is not an actual OS like Windows but an interface-like system that works with Linux-based operating systems (O'Kane, 2014).

While Hector SLAM provides robust mapping and localization, its primary output is a geometric map. For some autonomous robotic applications, understanding the objects within the environment is required. This setup can be used in hazardous indoor areas with different custom robots for exploration purposes. The custom-built robot can have any 2D LiDAR sensor that can publish information. In 2018, researchers compared various SLAM techniques with and without camera information. They categorized them into (a) 2D lidar-based: GMapping, Hector SLAM, Cartographer; (b) monocular camera-based; and (c) stereo camera-based. In their comparative study, Filipenko and Afanasyev (2018) tested three 2D LiDAR SLAM systems: GMapping, Hector SLAM, and Cartographer. They reported that GMapping did not provide reliable results. A key finding in their analysis was the comparison between the trajectories generated by Hector SLAM and Cartographer. Their evaluation using Absolute Trajectory Error (ATE) metrics showed that the Root Mean Square Error (RMSE) between these two trajectories was less than 3 cm. This led them to conclude that both systems produce practically the same results. In our work, we focus on Hector SLAM, which falls under the category of 2D LiDAR-based techniques. It stands out due to its low resource consumption and the fact that it does not require odometry. Hector mapping costs less CPU than other mapping techniques such as gmapping and cartographer. Hector SLAM depends only on sensor information to create a map, while gmapping requires additional odometry information. Another popular SLAM technique, Cartographer, requires at least 16GB of RAM (Filipenko and Afanasyev, 2018).

SLAM is sufficient for gathering information about the indoor area, but the addition of a camera to the system allows for gathering even more data and a better understanding of the area. The LiDAR sensor, which is one of the most utilized sensors, allows for the creation of a detailed map of its surroundings and the avoidance of obstacles in real-time. The robot can accurately navigate its environment with the help of the LiDAR sensor by using the ROS navigation package and SLAM techniques (Hess et al., 2016). Moreover, the camera can be used as an extra sensor in the system to improve SLAM (Chghaf, 2022, Sapmaz, 2022). In 2015, Kamarudin and his colleagues compared popular SLAM techniques with and without a camera. One of the interesting findings is that a camera without using a sensor failed the SLAM process; however, a sensor without a camera succeeded in performing it. Moreover, when both the camera and the sensor are utilized for performing SLAM, the map becomes more detailed. The work concludes that the camera may improve SLAM performance (Kamarudin et al., 2015). Another approach to camera usage with SLAM may be utilized to recognize objects with computer vision simultaneously but

as a separate process. This approach allows the camera to independently identify objects while SLAM focuses on map creation and robot localization. In 2019, Kashi and his colleagues developed an algorithm using gmapping (a SLAM technique) and a computer vision algorithm to create a map with recognized objects simultaneously. The camera has not been utilized directly to create a map but only to recognize objects (Kashi, Sriram, and Mohan, 2019). On the other hand, Hector SLAM is constrained with this approach since it cannot work with Python 3-dependent OpenCV libraries on ROS Kinetic (Fairchild and Harman, 2017) but only with the "find_object_2d" ROS package. As a result of this constraint, other methods have been developed by researchers. One of the methods is asynchronous, with map creation and image processing being performed independently or at different times. In the literature, utilizing Python 3-dependent OpenCV libraries with the Hector SLAM algorithm has been implemented by Hakim and his colleague in 2019. The object recognition has been implemented with YOLO (Fadhil and Hakim, 2019).

In this paper, an approach has been developed as we have not been interested in using the mentioned ROS package as a source of object recognition. Also, it has been applied to a custom-made autonomous robot whose motion has been provided by the navigation stack. The localization of the objects relies on the correct calculations of transformation and rotation matrices. Hector SLAM focuses on processing the map and implicitly navigating the robot as the camera takes pictures and stores them for processing purposes.

This paper presents an approach that combines Hector SLAM and computer vision using YOLO to detect objects and visualize them on another map on ROS Kinetic. We developed a custom mobile robot entirely from the scratch up and successfully deployed our algorithm on it. The robot is built using cost-effective reliable sensors and low-PWM, high-torque DC motors, providing a robust platform for experimentation. A Raspberry Pi 3B+ and an Arduino serve as the primary microcontrollers, interfaced via USB and integrated through a Robot Operating System (ROS) communication protocol, the details of which are discussed later in this paper. A key finding of our work is the successful demonstration of object detection and map generation without the need for odometry data.

2. METHODOLOGY

2.1. Hardware and Mechanical Design

The mobility The mobility of the custom mobile robot is provided by two differential wheels and one caster wheel. The robot consists of two lightweight wooden chassis. The lower part includes an L298N motor driver, an Arduino UNO microcontroller, and a power bank. The upper part consists of 8 batteries, a Raspberry Pi 3B+, a camera, and a LIDAR sensor. We chose the Raspberry Pi 3B+ model over the 4 because it is more cost-effective and sufficient for our algorithm's demands. Its primary function is to publish sensor data and images as a ROS stream. Due to its 1GB SDRAM, it is incapable of executing the algorithm simultaneously on its own and requires a connection to a more powerful machine for processing. However, it is possible to create communication through both 2.4GHz and 5GHz WiFi networks. The communication between the Raspberry Pi and the Ubuntu laptop was provided over the same IP. The Raspberry Pi is powered by an 11500 mAh power bank with a constant 2A output ampere and 5V output voltage, which is connected to the Pi using a USB cable. The Arduino and LIDAR are powered by the Raspberry Pi using compatible USB cables. The DC motors are connected to the output pins of the L298N motor driver, and the input pins of the motor driver are connected to the Arduino. The motor driver is powered by 8 pieces of 1.1V batteries in series (Fig. 1).

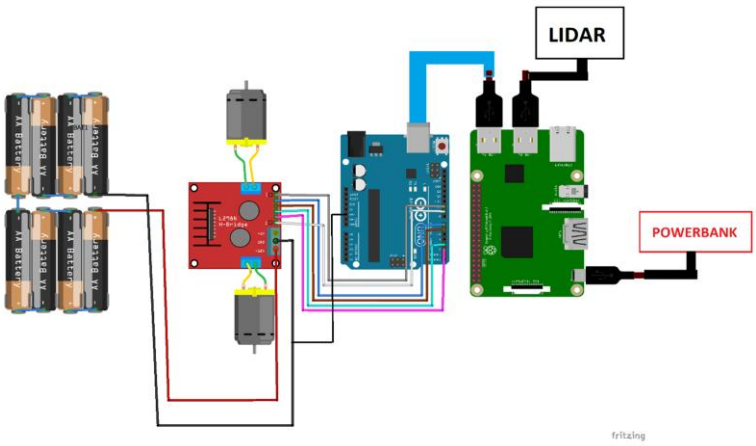


Figure 1:
Connection diagram of the mobile robot.

Two parts of the robot are designed on a CAD program for purposes of both simulation and real-life application. The chassis is manufactured using 3mm thick lightwood material as it is easy to cause necessary deformations without breaking it and is lighter than PLA material, which is generally used for 3D printers. The holder of the DC motor is an L-shaped connector that links the motor to the chassis. As extra deformation is not required, PLA material is used. The most important part of the mobile robot is the placement of the LiDAR sensor, as its 0th degree indicates the x-axis for the algorithm to work. The axes of the camera and LiDAR should be arranged to be in the same orientation. Therefore, the LiDAR was mounted in the same direction as the camera.

2.2. Software

A Raspberry Pi with Raspbian Stretch OS and a laptop with Ubuntu OS have been used to operate ROS Kinetic. The codes inside the Raspberry Pi were executed using the SSH protocol, which allows for remote control without the need for an external monitor to access its terminal. Performing SLAM and taking pictures simultaneously has been processed on the Ubuntu laptop. The Raspberry Pi has been set to be a ROS Master provider. ROS packages for the camera, sensor, and Arduino have been installed in the Raspberry Pi's ROS workspace to convert raw data into ROS topics. The connection between Arduino and ROS Master is provided to control the mobility of the robot. Similarly, relevant ROS packages such as Hector SLAM and the navigation stack have been installed on the laptop for SLAM processes.

RViz is an application that visualizes sensor data with its own tools, which can publish ROS topics or subscribe to existing ROS topics (Hyeong et al., 2015). The application creates a map by subscribing to ROS topics that the Hector SLAM technique publishes as the robot moves indoors. The navigation tool of RViz can be used to point to anywhere on the map and publish desired orientation information for the robot, since the motion depends on the navigation stack. The computer vision algorithm YOLO can only be used with Python 3 and OpenCV libraries. A separate virtual environment was created on the Ubuntu laptop that depends on Python 3 to perform object recognition. Object recognition has been carried out in a virtual environment with Python 3 installed using a computer vision algorithm. YOLO is designed to detect objects at three different scales—small, medium, and large—with respective scales of 32, 16, and 8 pixels. This allows the algorithm to detect objects of varying sizes with high accuracy (Bisallah et al., 2016).

2.3. Algorithm

A connection is established between an Ubuntu 16.04 laptop and a Raspberry Pi over the same IP address. The IP addresses are specified on two lines within the .bashrc files for both machines. On the Raspberry Pi's .bashrc file, both IP values are set to the Raspberry Pi's IP address. The Ubuntu laptop has a similar .bashrc file. The Master IP is set to the master's IP address, and the other IP value is set to the Ubuntu laptop's IP address. Then, both files are saved, and the source command is used in the terminal to apply these values permanently. This technique enables these devices to communicate over one ROS master.

The algorithm consists of two main launch files and four Python codes. Fig. 2 shows the brief algorithm scheme of these codes. Raspberry Pi 3B+ has 1GB of SDRAM, and it is not possible to process this data quickly every time due this limited memory. Therefore, a ".launch" file has been written for the Raspberry Pi 3B+. Launch files are ".xml" type files that include one or more executable nodes. The first launch file is executed on the Raspberry Pi and consists of three main tasks: publishing sensor stream, publishing camera stream, and establishing communication between Arduino and ROS. It is executed using SSH through the laptop. The second launch file on the laptop, which has three main tasks, is executed using the terminal of Ubuntu. These tasks are starting the RViz application with pre-configuration, subscribing to the sensor stream and executing both the Hector SLAM, and the navigation stack. It subscribes to ROS topics published by SLAM and the sensor stream, and publishing the required velocity.

The navigation is provided by an external node called "explore_lite." The mobile robot moves until all frontiers are cleared. This node subscribes to "map_msgs/OccupancyGridUpdate" and "nav_msgs/OccupancyGrid" messages published by Hector SLAM. These messages contain information about the updated and the full map. Then, the "explore_lite" node defines the direction of movement based on the map and its updates. For the mobile robot's movement, the "cmd_vel" topic is passed to the "move_base" topic. Simultaneously, the Arduino subscribes to the "cmd_vel" topic and uses the "geometry_msgs/Twist" message data to set the PWM values for the DC motors. Meanwhile, "move_base" utilizes the "cmd_vel" commands to plan and control the robot's movement for navigation. Finally, "move_base" publishes the "geometry_msgs/PolygonStamped" message that is utilized by "explore_lite" to complete the cycle of exploration.

The LiDAR sensor publishes a "scan" topic that contains information on 360 distance measurements per 1 degree. The Pi camera streams an "image_raw" ROS topic. The relations between the frames of the independent parts of the robot and the map are published by the "tf" topic. The position of the robot is established by the "slam_out_pose" topic. Our algorithm starts by subscribing to these ROS topics. The main goal is to find the position of any obstacles in the real-world frame. The "tf" ROS topic is used to track frames between the robot's chassis, "base_link," and the "map" itself. It broadcasts information about translation in x, y, z coordinates (in meters) and rotation (in quaternions) between specified frames. This information is used to determine the possible position of obstacles by using transformation matrices combined with the distance of the obstacle from the 'scan' topic. We positioned the sensor's zeroth degree to face forward and defined our area of interest as -10o to 10o. Our aim is to find the minimal distance and its corresponding angle relative to the robot's base, as published by the 'scan' topic within this specified area. The acceptable distance range for obstacle detection was determined to be between 0.4 and 1 meter through trial and error. This range was chosen because the photo quality degraded at closer ranges than 0.4 meters, and the number of detected objects increased at distances beyond 1 meter. Once an obstacle falls within the 0.4 to 1 m range, the algorithm identifies the minimum distance and its corresponding angle. These define the possible obstacle's position with respect to the robot's frame. This information is then conveyed into transformation matrices to acquire the obstacle's position with respect to the map. The main aim is to determine the position of the robot on the map frame, which is initially unknown.

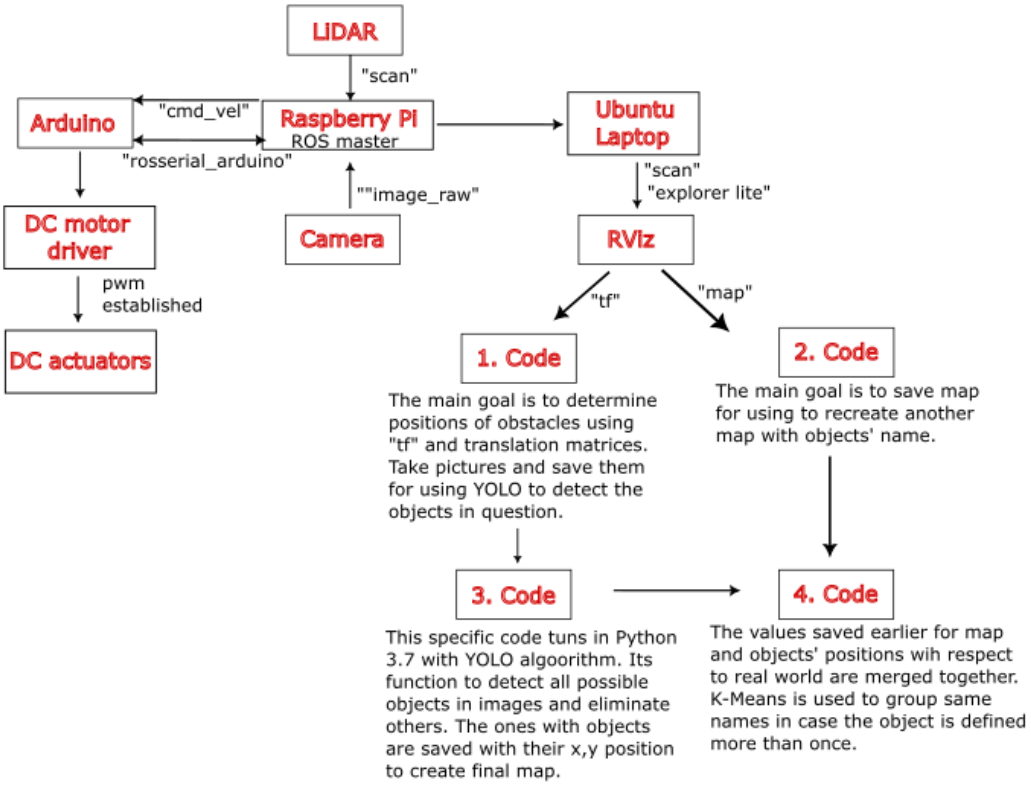


Figure 2:

The brief algorithm scheme illustrates connections between ROS and mechanical design.

The x-axis of the robot frame depends on how the LiDAR is mounted. The first and second elements of the measurement array define the x-axis and y-axis of the robot, respectively. The frame of the robot moves on RViz with respect to the map frame as the robot roams around. The x-axis of the map frame is defined as the x-axis of the robot's initial orientation when the SLAM process begins, and this frame remains fixed until the end of the process (Fig. 3).

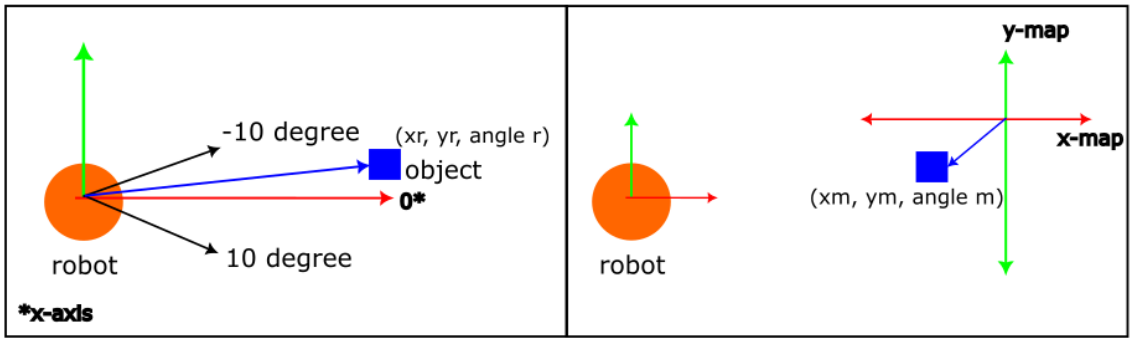


Figure 3:

The logic behind transformation matrices.

The first step is to locate the obstacle on the robot frame as a pair coordinate of (xr, yr),

$$x_r = d * \cos(\theta_r) \quad (1)$$

$$y_r = d * \sin(\theta_r) \quad (2)$$

where the obstacle distance (d) is defined by the minimum value among the collected 20 samples from of LiDAR in range -10 to 10 degrees. The angular resolution of the sensor is 1 degree, and it can collect 360 measurements. The x-axis of the LiDAR is assumed to be 0 degrees. Using the right-hand rule, degrees between -10 and 10 can be determined. We have limited the data collection by not processing each measurement. The angle of an object with respect to the robot frame is a value between -10 and 10, represented as θ_r. If θ_r is between -10 and -1 degrees, yr is multiplied by -1 for equation (2) as y point should be exact negative.

The next step is to find where the robot is located with respect to the map frame. As the robot frame translates and rotates on the RVIZ screen, unique transformation matrices are published by the ROS tf package for each frame. The transformation matrix quantifies the displacement of the robot frame relative to the map frame due to the robot's motion. This matrix contains information about the amount of translation and rotation of the robot frame relative to the map frame. In this step, the values of xr, yr have been calculated. Pair of (xt, yt) and the amount of rotation φ (in quaternion) are published by "tf" ROS topic. The first step is to apply the rotation matrix to the known xr, yr points.

$$R = \begin{bmatrix} cos\varphi & -sin\varphi & 0 \\ sin\varphi & cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ 1 \end{bmatrix} \tag{3}$$

$$R = \begin{bmatrix} cos\varphi * x_r - sin\varphi * y_r \\ sin\varphi * x_r + cos * y_r \\ 1 \end{bmatrix} \tag{4}$$

where R defines the position of obstacle in rotated frame amount of φ degree.

The next step is to apply matrix T to R (Craig, 2014)

$$T = \begin{bmatrix} 1 & 0 & -x_t \\ 0 & 1 & -y_t \\ 0 & 0 & 1 \end{bmatrix} . \boldsymbol{R} \tag{5}$$

$$T = \begin{bmatrix} cos\varphi * x_r - sin\varphi * y_r - x_t \\ sin\varphi * x_r + cos * y_r - x_t \\ 1 \end{bmatrix} \tag{6}$$

where T matrix represent the robot location with respect to the map frame which are xm and ym. Equations 1-4 are embedded into the algorithm to determine the possible positions of obstacles. This pair of (xm, ym) coordinates is saved into a pickle file with image names to process later. As soon as the LiDAR sensor detects an obstacle within the specified distance range (0.4m to 1m), an image is captured. Prior to any processing, the image data from the 'image_raw' ROS topic is converted into a Computer Vision (cv) format, which is compatible with Python's image processing libraries. The second Python code is activated after the exploration process is complete. Its aim is to save the final map as a list of (x, y) coordinate pairs within a pickle file. The 'map' ROS topic provides information about the final map as a 1024x1024 matrix. This matrix uses the following values: 0 for explored and unoccupied areas, -1 for unexplored areas, and 1 for explored and occupied areas. This code specifically extracts the (x, y) coordinates corresponding to the value '1' to create a usable final map representation for the next processing. For a more detailed workflow, please refer to supplementary material 1.

The third and fourth Python codes were executed with Python 3.7 (Fig. 4). To run these codes, a virtual environment was set up on Ubuntu 16.04, and this virtual environment was made compatible with Python 3.7. The third Python code executes an object recognition algorithm

(YOLO) by filtering each image recorded in the first Python code one by one. Within this code, first, the 'pickle' file obtained from the first code is opened. Inside the 'pickle' file, there are x, y coordinates and file paths. The images within the file paths are sent to the method where the image processing algorithm starts, along with their x and y coordinates. For each photograph sent, if an object is detected in the photograph, the x and y coordinates corresponding to the file path and the names of the objects are added to separate matrices. If no object is detected, no action is taken. All matrices are first saved in dictionary format. This dictionary is then converted back into a 'pickle' file. The fourth Python code, on the other hand, is the code where the map information coming from RViz is combined with the object names and object coordinate information coming from the algorithm. Additionally, since the same object can be detected multiple times, it performs cluster grouping for a single object using k-means. By definition, the k-means clustering algorithm groups data points into clusters by minimizing the distance between each point and its cluster's centroid.

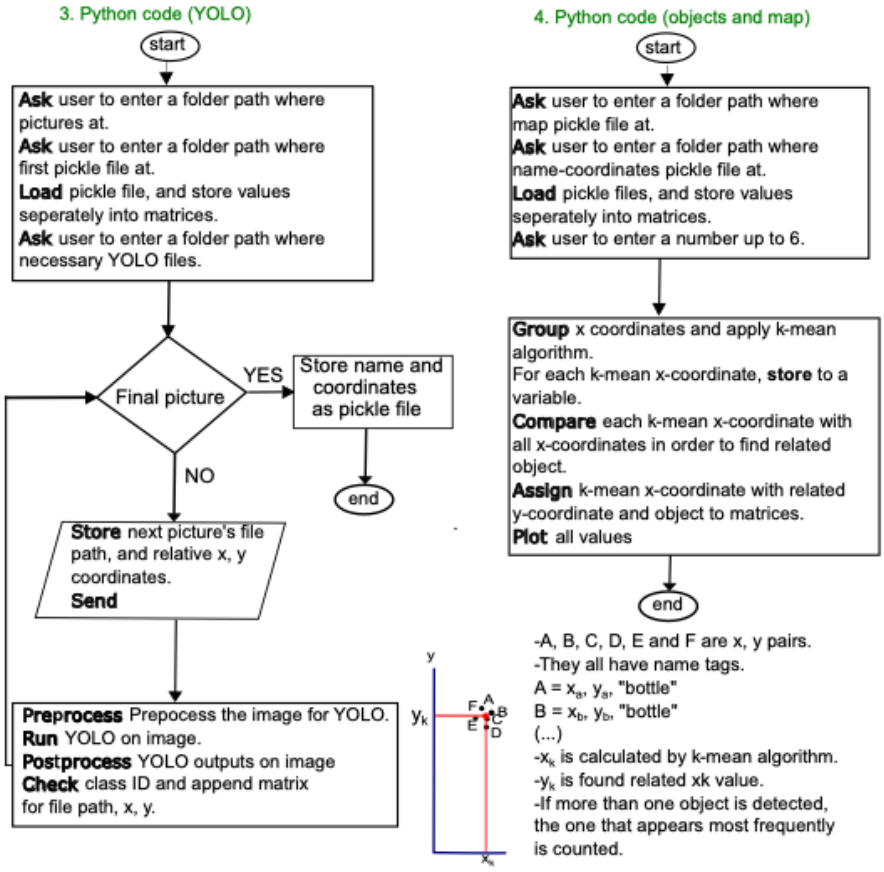


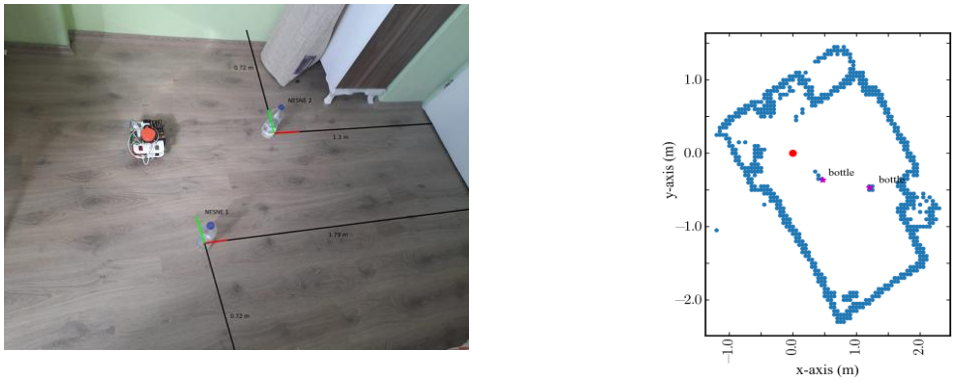
Figure 4:
The flowchart of third and fourth Python codes.

3. RESULTS

The algorithm has been tested in two different places that were accurately recognized 18 of 20 objects, resulting in a detection accuracy of 90 %. Positioning error was calculated comparing points that the algorithm had returned with real x, y (m) values which were distances of objects' point (0, 0) to nearest wall/obstacle. For area calculation, we used the longest dimensions of the regions, treating them as rectangular approximations. This method was applied to both the map

obtained in RViz and the real-life areas. Each object has its imaginary frame. The false positioning of the objects is calculated as pair of x, y %, and overall result is calculated as 2.2%, 5.14%.

In the 1st room, two bottles were recognized correctly. The room, and a typical map created by the algorithm can be seen in Fig. 5. Blue dots represent obstacles and objects. Additionally, purple dot is the object detected by algorithm, and red dot shows reference point (0,0) of the map.



a. *b.*

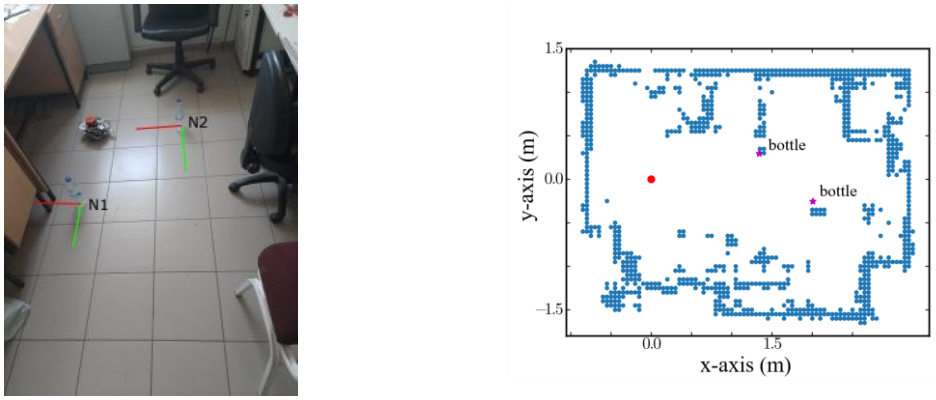
Figure 5:
The first room with bottles and its final map;
a. *The indoor* ***b.*** *The map*

Table 1 shows the x and y values calculated by the algorithm and ground truth (exact) observations. For the first object, the exact coordinates are (1.79, 0.72) m, and for the second object, they are (1.3, 0.72) m. The accuracy of the x and y values for the first object are 97.02% and 90.8%, respectively. For the second object, it is 96.82% for x and 95.1% for y.

Tablo 1. Distance errors for the 1st room.

Case	Object 1 (x, y) m	Object 1 Error x%, y%	Object 2 (x, y) m	Object 2 Error x%, y%
1	(1.75, 0.78)	2.2, 8.3	(1.3, 0.72)	0, 0
2	(1.76, 0.64)	1.6, 11.1	(1.23, 0.76)	5.3, 8.3
3	(1.66 , 0.67)	7.2, 6.9	(1.34 , 0,67)	3, 6.9
4	(1.76, 0.62)	1.7, 13.9	(1.23, 0.72)	5.3, 6.9
5	(1.75, 0.68)	2.2, 5.8	(1.269, 0.738)	2.3, 2.4
The ground truth (true) location of the the 1st object is (1.79, 0.72) m, and 2nd object is (1.3, 0.72) m.				

Furthermore, the identification success of the algorithm is 80 % for the second room. The 2nd room and a typical map created by the algorithm is given in Fig. 6.



a. *b.*

Figure 6:
The second room with bottles, books and its final map;
a. *The indoor* ***b.*** *The map*

The results observed for the 2nd room are summarized in table 2. The ground truth location of the first and second objects are (0.94, 2.1) m and (1.2, 1.2) m, respectively. The average identification scores for the first and second object are 97.59% and 94.86%, respectively.

Table 2. Identification scores of the algoritm observed for the 2nd room.

Case	Object 1 (x, y) m	Object 1 Error x%, y%	Object 2 (x, y) m	Object 2 Error x%, y%
1	(0.95, 2.13)	1, 1.6	(1.52, 1.18)	26.6, 1.6
2	(0.94, 2.13)	0, 1.6	(1.2, 1.3)	0, 8.3
3	(- , -)	-, -	(1.18, 1.15)	1.6, 4.16
4	(-, -)	-, -	(1.23, 1.19)	2.5, 0.8
5	(0.88, 2.18)	6.38, 3.8	(1.14, 1.21)	5, 0.83
The ground truth (true) location of the the 1st object is (0.94, 2.1) m, and the 2nd object is (1.2, 1.2) m.				

The overall MSE scores for the x and y coordinates found by the algorithm across 18 takes are 0.00855 and 0.00271, respectively.

4. DISCUSSION

This research presents a cost-effective mobile robot for mapping, object localization, and detection. We have deployed a wide variety of tools such as the LIDAR sensor, actuators, a high-resolution camera, motors, Arduino MPU, and Raspberry Pi CPU. The communication between hardware components was performed using ROS on the open-source Ubuntu operating system. The investigations were conducted at two sites, and the following results were obtained.

In the first stage, we investigated how well the SLAM algorithm can generate the actual maps, visualize them, and compute their area. The overall results obtained in our study are as follows. First, the algorithm can compute the area of the first and second rooms with high scores of 4.49% and 4.48%. In the second stage, the goal was to identify how well the algorithm can detect the objects. The results show that the average correct identification of the objects is 85%. The slight errors could be due to image resolution and lighting. Finally, we investigated whether the

proposed approach could determine the location of the object as an addition to the detection process. The results showed that the approach could locate the object with an average performance of 95.5%. We expect that the errors between the ground truth and the values computed by the algorithm may be due to the sensitivity of the LIDAR.

In their 2024 work, Chowdhury and Houssain implemented Hector SLAM for indoor navigation with obstacles, using a camera to capture obstacle images (Chowdhury and Houssain, 2024). While successful in rectangular environments, their approach was not tested with randomly shaped places, and they did not incorporate obstacle mapping as we have done in our study. Furthermore, unlike their method which utilized an additional IMU for position correction, we chose not to implement extra odometric information for defining the robot's and obstacle's positions. In their 2019 paper, Kashi and his team developed an interesting algorithm that used Gmapping (a SLAM method) along with computer vision to build a map while also identifying objects. Interestingly, much like our own study (Kashi, Sriram, and Mohan, 2019), they didn't use the camera to directly create the map, but specifically for recognizing objects. They successfully took pictures of obstacles and labeled them in Rviz. They also used a hall effect sensor to get the robot's position, as Gmapping needs external odometry data. However, their testing was limited to just one environment. Also, they didn't include the actual positions of the obstacles, either in Rviz or real-world data, for comparison. Their main focus was on how confident they were in identifying the objects. In the literature, utilizing Python 3-dependent OpenCV libraries with the Hector SLAM algorithm has been implemented by Hakim and his colleague in 2019. The object recognition has been implemented with YOLO. They developed a smart stick for people with impaired vision. They created a map of the environment using Hector SLAM and then applied a computer vision algorithm to recognize objects. The objective is to establish a pathway that would let individuals navigate without encountering obstacles and provide them with information about their environment (Fadhil and Hakim, 2019). However, they haven’t developed their algorithm using an autonomous mobile robot and haven’t taken pictures simultaneously as their stick moves around. Different from the mentioned studies, we include object identification processes, and the results are promising. We expect that the methodology we suggest can be applied in various applications, such as object identification and localization in hazardous areas where humans cannot access.

Although the performance of the approach is reasonable, this study is not without limitations. Firstly, the investigation was carried out in indoor rooms with sunlight. Future studies could extend the approach to sites with limited lighting. Secondly, the scanning time duration for a 4 m^2 indoor room is approximately 10 minutes with this version of the robot. This duration could be shortened by modifying the hardware, software, and mechanical design. Thirdly, the approach was tested in a human living room. Additional studies could be implemented to investigate its sensitivity in hazardous areas. The current version of the robot operates on a Wi-Fi connection, whose sensitivity can decrease with distance. To localize sites remotely over greater distances, the connection could be made via radio communication. The transformation and rotation matrices are implemented correctly in the system, as the error values remain consistent, with only a few variations observed in certain results. Moreover, the navigation stack is utilized with custom-made values to navigate the system, ensuring unique parameters for each project. Since it is a frame-based system, the replacement of the laser scanner holds significant importance. In this paper, to simplify the process, the x-axis of the laser scanner has been selected, and all calculations are performed relative to this specific axis.

5. CONCLUSION

In this study, a mobile robot was built with the ability to move on its own, recognize objects, and understand its surroundings. The robot's control and computing parts communicated in real time using the Robot Operating System (ROS). It used the YOLO (You Only Look Once) method

to recognize objects and SLAM (Simultaneous Localization and Mapping) to figure out where it was and to map the area around it. During testing, it was noticed that sometimes the robot finished mapping before it could detect certain objects. To solve this, an interactive tool was used so users could click on a location, and the robot would move there on its own to take pictures.

In conclusion, this robot offers a budget-friendly alternative to expensive commercial systems, and it can perform important tasks like mapping, finding its location, and recognizing objects. Thanks to Hector SLAM, it can accurately figure out where it is without needing extra location tools. The results show that this method could work well for indoor robot tasks. Future work will focus on testing the robot in more challenging and risky real-world settings.

CONFLICT OF INTEREST

The author(s) confirm that there are no known conflicts of interest or shared interests with any institution, organization, or individual.

CONTRIBUTIONS OF AUTHORS

Saran Sapmaz data collection, data analysis, and writing the article, Mahmut Pekedis workflow design and data visualization.

REFERENCES

1. Bisallah, H.I., Oguin, K.J. and Oguine, O.C. (2022), YOLO v3: Visual and Real-Time Object Detection Model for Smart Surveillance Systems(3s), doi: 10.48550/arXiv.2209.12447
2. Chghaf, M.R. (2022), Camera, lidar and multi-modal SLAM systems for autonomous ground vehicles: a survey, *Journal of Intelligent & Robotic Systems,* volume (105):2. doi: 10.1007/s10846-022-01582-8
3. Craig, J.J. (2014), *Introduction to Robotics Mechanics and Control (3 ed.),* Leeds: Pearson Education Limited
4. Fadhil A. and Hakim H. (2019) Indoor low cost assistive device using 2D SLAM based on LIDAR for visually impaired people, *Iraqi Journal of Electrical and Electronic Engineering* (15)2: pp 115-12, doi: 10.37917/IJEEE.15.2.12
5. Hess, W., Kohler, D., Rapp, H. and Andor, D. (2016), Real-Time loop closure in 2D LIDAR SLAM, *2016 IEEE International Conference on Robotics and Automation (ICRA),* pp: 1271-1278., doi: 10.1109/ICRA.2016.7487258.
6. Hossain, A. and Chowdhury, R.H. (2024) "Implementation of Hector SLAM Algorithm for Mapping Indoor Environments with Obstacles," *2024 IEEE 3rd International Conference on*.
7. https://jokane.net/agitr/agitr-letter.pdf, Date of Access: 06.06.2023, Topic: A Gentle Introduction to ROS
8. https://sceweb.sce.uhcl.edu/harman/CENG5437_MobileRobots/Webitems2020/ROS_ROBOTICS_BY_EXAMPLE_SECOND_EDITION.pdf, Date of Access: 07.06.2023, Topic: *ROS Robotics By Example-Second Edition. Leeds: Packt.*
9. Hyeong, R.K., Lee, S., Park, T. and Kim, C. (2015) RViz: a toolkit for real domain data visualization, *Telecommunication Systems* (60): 337-345, doi: 10.1007/s11235-015-0034-5
10. K. Kamarudin *et al*. (2015), Improving performance of 2D SLAM methods by complementing Kinect with laser scanner, *2015 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*, Langkawi, 278-283, doi: 10.1109/IRIS.2015.7451625.
11. Kashi, M.S, Sriram, T.B. and Mohan, R. (2019), An Approach to labelled indoor mapping using SLAM and object detection. *2019 IEEE Region 10 Symposium (TENSYMP)*, 321-325. doi: 10.1109/TENSYMP46218.2019.8971298

12. Kohlbrecher, S., Meyer, J., Stryk O. von and U. Klingauf (2011), A flexible and scalable SLAM system with full 3D motion estimation, *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR),* 155-160, doi: 10.1109/SSRR.2011.6106777.

13. M. Filipenko and I. Afanasyev (2018). Comparison of Various SLAM Systems for Mobile Robot in an Indoor Environment. In 2018 International Conference on Intelligent Systems (IS), pp. 400-407. doi: 10.1109/IS.2018.8710464.

14. Marder-Eppstein, E. et al. (2010), The office marathon: robust navigation in an indoor office environment, *International Conference on Robotics and Automation*, 300-307, doi: 10.1109/ROBOT.2010.5509725

15. *Robotics, Automation, Artificial-Intelligence and Internet-of-Things (RAAICON)*, Dhaka, Bangladesh, 2024, pp. 218-223, doi: 10.1109/RAAICON64172.2024.10928605 https://dspace.mit.edu/bitstream/handle/1721.1/119149/16-412j-spring-2005/contents/projects/1aslam_blas_repo.pdf, Date of Access: 07.06.2023, Topic: *Blas MR and Riisgaard S (2005) SLAM for Dummies: A Tutorial Approach to Simultaneous Localization and Mapping*.

16. Sapmaz, S. (2022), Eş zamanlı konum belirleme ve haritalama için bilgisayarlı görü ve sensör tabanlı otonom gezgin robot [A computer vision and sensor based autonomous mobile robot for simultenaous localization and mapping], *M. Sc. Thesis*, Ege University, TR.