FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Information Technology Course Module Autonomous Intelligent System
By Prof. Dr. Peter Nauth

Kidnapped Robot Problem Solving with Object Detection Using YOLO and AMCL Algorithm in Simulated Environment

Mashnunul Huq
Matriculation No: 1384042
mashnunul.huq@stud.fra-uas.de

Sourav Paul Sumit
Matriculation No: 1344118
sourav.sumit@stud.fra-uas.de

Md. Tanzeem Hasan Mahmud
Matriculation No: 1364198
mahmud.mdtanzeemhasan@stud.fra-uas.de

***Abstract*: This paper represents an improved algorithm to solve a robotic conundrum where the robot is restarted in an unknown place (kidnapped position) and recognize the current position to start the basic function recognizing the map and exact position of the robot. The proposed solution utilizes YOLO object detection to recognize landmarks and corresponding initial position for estimating the current position using AMCL algorithm based on the landmarks. This experiment proves to have high accuracy and robustness in solving the kidnapped robot problem in complex and micro-environments. This experiment contributes to the advancement of mobile robot localization and navigation with potential applications in autonomous diving, warehouse logistics and search and rescue operations.**

Keywords—Adaptive Monte Carlo Localization, Simultaneous Localization and Mapping, You Only Look Once, Kidnapped Robot, Simulation, Light Detection and Ranging Sensor, Depth Camera Sensor.

I. INTRODUCTION

The Kidnapped Robot problem is a well-known challenge in the field of Autonomous Mobile Vehicle Devices. It occurs when a robot is suddenly moved to an unknown location without any prior knowledge of its new position and orientation by external forces like heavy wind challenge, earth quake or deliberate attacks, software malfunctions or at shutdown moved by human intervention. The problem is particularly relevant for autonomous robots that rely on mapping and localization techniques to navigate and perform tasks in complex environments.

Kidnapped Robot problem presents a significant challenge for autonomous robots as their previous map and localization information becomes obsolete when they are moved to a new location. Even if it occurs inside a prebuilt map area the robot's SLAM (Simultaneous Localization and Mapping) gets affected by the miscalculation of the current position and the goal position because of the unknown initial position measurements. Without accurate localization information, the robot may become lost, unable to navigate to its target location, or even collide with obstacles. This can lead to a range of practical problems, including damage to the robot, loss of valuable data, and delays in completing tasks.

Several approaches have been experimented in solving kidnapped robot problem including feature-based localization, Monte Carlo localization, and particle filter methods. However, these methods require significant amount of computational resources and sometimes inopportune with mobile robotic system and time as most of the robotic system uses raspberry pi or arduino boards with weak computational chipsets.

In recent years deep learning techniques based on OpenCV library have shown great potential in addressing a solution to the kidnapped robot problem. Specially, object detection algorithms such as YOLO (You Only Look Once) algorithm can recognize landmarks in an unknown environment being useful to estimate robot's initial position and orientation to support prebuilt algorithms like AMCL for faster localization with less computational power and time. Time required for AMCL to detect current position at restarting situation with depth camera and LIDAR sensor without additional object detection is huge as the robot requires certain amount of movement for data collection.

In this experiment, we propose a novel approach to solving kidnapped robot problem using YOLO object detection and AMCL algorithm. We demonstrate the effectiveness of our approach through series of experiments in robust house environments which can also be symbolic to a warehouse or a departmental store with multiple rooms and places. Different products detected in different rooms can also be symbolic to separated floor levels and store management systems like amazon ware house or rewe departmental store.

II. THEORITICAL APPROACH

A. Sensors used

Though we have experimented in simulation environment but we also have tested physical behavior of two sensors. We also have used these sensors in the simulation provided in gazebo from the manufacturing companies.

1. Intel Realsense D435:

The Depth Camera D435i is a camera device of Intel Real-sense technology used to capture objects or

environments 3D images in real-time and the depth data. This stereo camera uses two infrared cameras and a RGB (3 channel) camera in combination to generate depth information that can be used in robotics, virtual reality as well as in augmented reality. The D435 features an Inertial Measurement Unit (IMU) which provides accurate data on the camera's movement in any situation by opening the door for rudimentary Simultaneous Localization and Mapping (SLAM). This allows the camera to accurately track its position in better point-cloud alignment and acceleration in three-dimensional space, making it ideal for applications such as robotics, augmented reality, and virtual reality. The D435 also includes Intel RealSense technology, which enables advanced depth sensing capabilities. Because of its compactness, lightweight, advanced features such as object tracking, gesture recognition, and scene reconstruction and easy to integrate into a variety of applications, it is now a popular choice for developers and researchers working in the field of computer vision to create immersive and interactive applications. A depth camera works with pixels which have a different numerical value associated with each numerical distance from the camera or depth. D435 camera have both an RGB and a depth system, which can give pixels with all four values or RGBD. The output from the depth camera can be displayed in Intel realsense viewer software with the color image shown side by side with the depth image, where each different color in the depth map represents a different distance from the camera. In this case, cyan is closest to the camera, and red is furthest.

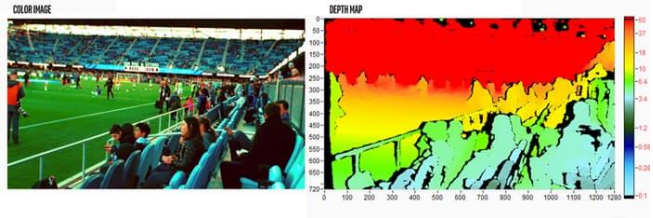

Figure 1: Picture depth calculation system in intel realsense D435

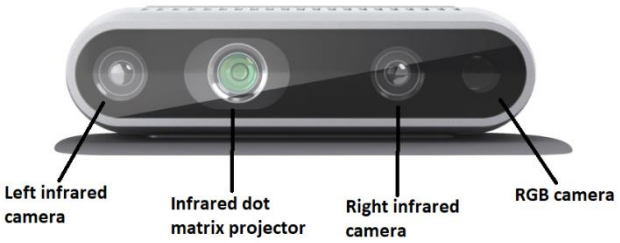

Figure 2: Intel Realsense D435 depth camera description

2. RPLIDAR:

RPLIDAR is designed as a 2D type laser scanner or Light Detection and Ranging sensor for using in robotics, autonomous vehicles, and other applications where extremely high accuracy and reliability is essential in the terms of sensing the environment and measure distances to surrounding objects of to create 2D maps. It has a rotating laser rangefinder, which scans the surrounding objects or the entire environment in a 360-degree field of view.

To measure distances of an object from the RPLidar sensor, it uses laser light in its field of view, and generates a 2D point cloud or map of the surrounding environment in real-time which can be used in obstacle detection, navigation or in other robotic applications when accurate mapping or object detection is required. RPLidar can detect objects up to a range of several meters, and can provide up to 10,000 samples per second, making it suitable for high-speed applications. After initializing communication by a host system, RPLidar communicate with its host by sending the binary data packet through Uniform packet formats. Basically, it cannot send any scanned data automatically except generating a request by the host. In order to start scanning operation of RPLidar, host system need to send a pre-defined request to RPLidar. Then it will initialize scanning operation and sends continuously the scanned data to the host system. (SLAMTEC, 2021)

Figure 3: RPLIDAR manufactured by RoboPeak

B. Convolutional Neural Network

Convolutional Neural Network is a computational algorithm mostly used in classification, object detection, and prediction. CNNs are designed to automatically extract features from input data, such as images, by using a series of convolutional layers. In these layers, a set of filters is applied to the input data, each filter detecting a specific feature or pattern. The output of each filter is then passed through a non-linear activation function, such as the Rectified Linear Unit (ReLU), which helps to introduce non-linearity into the network and improve its ability to learn complex patterns. The output of the convolutional layers is then typically fed into one or more fully connected layers, which use the extracted features to make predictions about the input data. For example, in image recognition tasks, the fully connected layers might output a probability distribution over different classes, indicating which class the input image most likely belongs to. CNNs are trained using a variant of backpropagation called stochastic gradient descent (SGD), which involves iteratively adjusting the weights and biases of the network to minimize a loss function. The loss function typically measures the difference between the predicted output of the network and the true output, and the goal of training is to find the set of weights and biases that minimize this difference. (Saha, 2018)

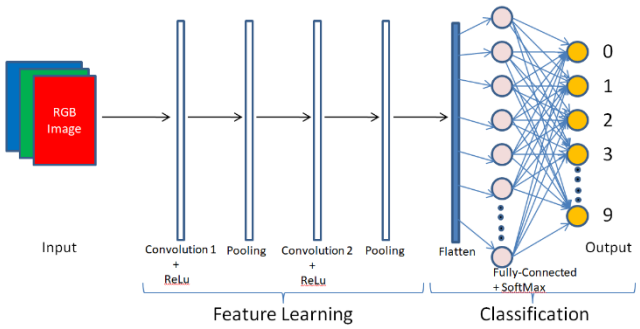

Figure 4: Layers of CNN

One of the key advantages of CNNs is their ability to automatically learn features from raw input data, without the need for manual feature engineering. This makes them well-suited to a wide range of computer vision tasks, such as object detection, image segmentation, and facial recognition.

C. Open Source Computer Vision Library

It is an open source library with comprehensive set of classic and state-of-the-art real-time image processing and machine learning algorithms to provide common infrastructure for the applications of computer vision. As well as to accelerate the use of machine perceptions with a range of tools and functions such as recognizing faces or scenery, detect the objects, image segmentation, tracking a moving objects, extracting three-dimensional objects, etc it is commonly used. It supports a wide range of programming languages such as C++, python, Java, MATLAB and it can be used on windows, Linux, macOS etc platforms. OpenCV supports various object detection and recognition techniques such as Haar cascades, HOG (Histogram of Oriented Gradients), and deep learning-based methods to support tasks like clustering, classification, and regression. OpenCV provides a range of image processing functions such as image filtering, edge detection, thresholding, and image segmentation.

D. Adaptive Monte Carlo Localization

AMCL or Adaptive Monte Carlo Localization is an algorithm used for the mobile robot localization process with high accuracy and reliability in an unstructured environment such as muddy, hill or in any hostile weather. The Adaptive Monte Carlo Localization algorithm is adapted from the Monte Carlo Localization algorithm to come out from some limitations of MCL algorithm. MCL algorithm basically used to represent the pose of the robot with particles on a two-dimensional grid map. It calculates the particles weight and then it determine the robot's estimated pose and locate the robot on the map. But in solving robot kidnapping difficulties, MCL fails localization if the robot poses changes. So AMCL algorithm throve to improve localization accuracy to extremely high.

The particle filter used in AMCL maintains a set of particles, where each particle represents a hypothesis of the robot's pose. The particles are sampled from a probability distribution that represents the robot's initial pose uncertainty. The particle filter then updates the weights of the particles based on the sensor measurements and the measurement model. The particles with higher weights represent more likely pose estimates. During the re-sampling, basically AMCL algorithm is an adds free particles. In this case, Kullback-Leibler divergence or in short form KLD is used in the Adaptive Monte Carlo Localization algorithm to resample particle. (Peng, Zheng, Lu, & Liao, 2018)

However, here calculating the free particles number based on long term estimated weights ωslow –

ωslow = ωslow + αslow (ωavg - ωslow),

and the short term estimated weights ωfast –

ωfast = ωfast + αfast (ωavg - ωfast)

Here, ωavg is all particles average weight, αslow is used to estimate the attenuation rate of the exponential filter with long-term and αfast for the short-term weight.

The required number of particles can be calculated based on the particle weights distribution.

The upper bound Ntop of the number of particles -

$$N_{\text{top}} = \frac{k-1}{2\alpha}\left(1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}\beta\right)^3 .$$

Here, α is the maximum error and β is the standard normal distribution quantile between the true distribution and the estimated distribution. k is the number of nonempty in the state space of the particle. [5]

The algorithm adaptively adjusts the number of particles in the filter based on the quality of the particle set. If the particle set becomes too diverse or sparse, the algorithm increases the number of particles to improve the localization accuracy. Similarly, if the particle set becomes too dense or correlated, the algorithm reduces the number of particles to improve computational efficiency.

One of the strengths of AMCL is its ability to handle different types of sensors, such as laser range finders, cameras, and sonar sensors, and integrate them in a probabilistic framework. The algorithm can also handle different types of noise and uncertainty in the sensor measurements.

E. You Only Look Once

YOLO is a real-time object detection algorithm that uses a single convolutional neural network (CNN) to simultaneously detect and classify objects in an image. The YOLO algorithm consists of two parts: a CNN and a post-processing step. The CNN takes an image as input and outputs a tensor (built on OpenCV) containing the probabilities of each class for each grid cell in the image. The CNN architecture used in YOLO is similar to other CNNs used in image classification, such as VGG and ResNet, but with modifications to allow for object detection.

The post-processing step in YOLO involves filtering the predictions made by the CNN and generating bounding boxes around the detected objects. This is done by applying non-maximum suppression (NMS) to the predicted boxes, which removes overlapping boxes and keeps only the box with the highest probability for each object. (Periwal, 2020)

One of the strengths of YOLO is its speed. YOLO can process images in real-time on a GPU (Nvidia), which makes it suitable for applications such as video surveillance, autonomous driving, and robotics. Another strength of YOLO is its ability to detect multiple objects in an image, even if they are of different sizes, scales, and aspect ratios.

F. Robot Operating System

ROS (Robot Operating System) is an open-source software framework for building robotics applications. ROS provides a set of tools and libraries for building and running robotics software, including drivers for sensors and actuators, algorithms for perception and control, and communication tools for distributed computing.

The main features of ROS include:

1. Nodes: ROS nodes are individual processes that perform specific tasks in a robotics system. Nodes can communicate with each other using a publish-subscribe messaging system or a request-response service.
2. Topics: ROS topics are named buses over which nodes exchange messages. A node can publish messages to a topic or subscribe to messages from a topic.
3. Services: ROS services allow nodes to request a specific task to be performed by another node and receive a response once the task is complete.
4. Parameter server: ROS provides a parameter server for storing and retrieving parameters that can be used by nodes during runtime.
5.Tools: ROS provides various tools for debugging, visualization, and simulation of robotics applications, such as RViz for visualizing sensor data and Gazebo for simulating robot models.

G. Turtlebot 3

This is an affordable, open-source, versatile and customizable popular robotics platform to easy use specially designed for learning and experimenting purposes. This third generation TurtleBot is designed implements in a variety of projects such as basic and advanced both robotics research and applications. By the collaboration between Robot Operating System (ROS) and Robotics, this robot platform and components and the component is created.
The TurtleBot 3 can be controlled remotely by the host device. It runs simultaneous localization and mapping or SLAM algorithm in its navigation. Through sensing with the equipped 360-degree Light Detection and Ranging sensor (Lidar), a camera such as Depth Camera D435i, and an Inertial Measurement Unit (IMU), which enables turtlebot 3 to navigate in various structured and unstructured environments with avoiding obstacles and mapping its surroundings. This robotic platform is based on a single-board computer, such as Raspberry Pi, and is powered by Robotic Operating System (ROS) which makes easier to develop and implement complex robotics algorithms and applications. We have used in our simulation the Burger type which is the most basic model for the learning and researching purpose

H. PR2 Robot

Personal Robot 2 (PR2) is a platform developed by Willow Garage for robotics research and development particularly in the areas of manipulation, perception, and cognition. The PR2 software system is written in Robot Operating System (ROS) entirely whereas PR2 capabilities are available via ROS interfaces. Including 1000+ software libraries the PR2 hardware platform enables researchers to focus on new capabilities. Because of the common platform it's easier to share the research results for the researchers in reproducible way. However, this robot can move and navigate human environments, and combines with the dexterity to grasp and then it can manipulate the objects in structured and unstructured both environments. Basically it is a mobile robot with two back-drivable arms, a torso, and a head that can move and sense the environment by using its cameras, lasers, and microphones. It has two 8-core servers included 24 Gigabytes of RAM each located in the robot base. After the last update of gazebo simulation software, teleoperation node of pr2 robot is not working properly but due to its robust structure and vast amount of basic working capabilities, pr2 robot simulation is suitable for simulation environments.

III. WORKING METHODOLOGY & ALGORITHM

A. Environment Consideration

A house or warehouse with six rooms, each containing a unique object, can be an interesting place to explore. The objects inside each room can range from simple to complex, and each room can hold its own mystery and intrigue through unique objects to be detected.
For instance, the first room may contain a grandfather clock, with its intricate gears and chimes. The second room may hold a collection of antique books, with their worn pages and intricate illustrations. The third room could be a music room, with a grand piano at its center, waiting for someone to play a beautiful melody. Moving on, the fourth room may be a workshop, with various tools and materials for crafting and creating. The fifth room could be a gym, with exercise equipment and weights to keep one healthy and fit. And finally, the sixth room may be a meditation room, with soft lighting and comfortable sofa cushions for relaxation and reflection.
For our case in simulation we chose to keep 6 unique objects which are already trained in yolov2-tiny.yaml file mentioned in the github repository. So we have chosen diningtable for room1, persons to be detected in room 2, sofa detected in room3, microwave oven detected in room4, suitcase in room5 and a fire hydrant in room6 as landmarks.

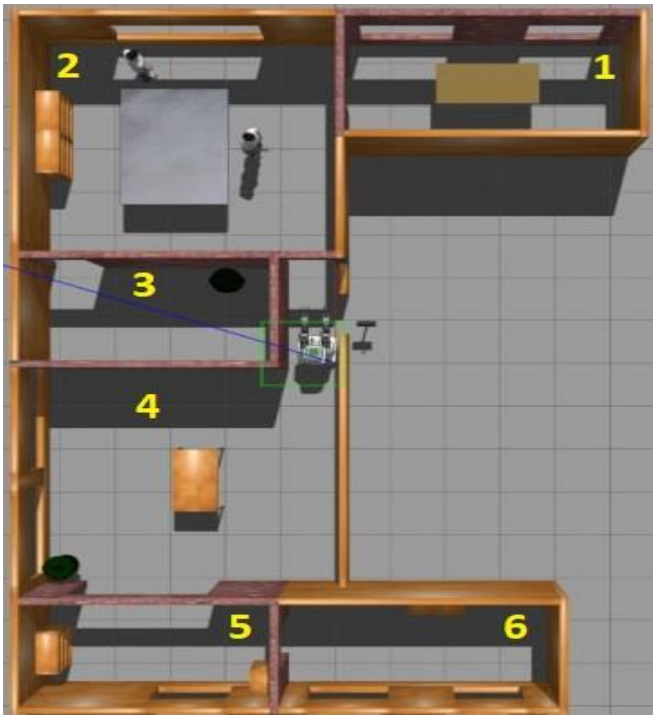

Figure 5: Different rooms in the house simulation file demarked.

Alternatively, in a warehouse setting, there could be multiple rooms dedicated to different products or materials. For instance, one room could hold electronics, while another

could hold textiles or furniture. There could also be rooms for storage, packing, and shipping, among others. In either cases, the six rooms or more in a warehouse provide ample space to organize and store different items or products. And in a house, they can serve different purposes and cater to different interests and hobbies of the occupants.

Overall, a house or warehouse with multiple rooms can be an exciting and diverse space to explore and detect easily by the robot when it gets restarted suddenly.

B. Algorithm

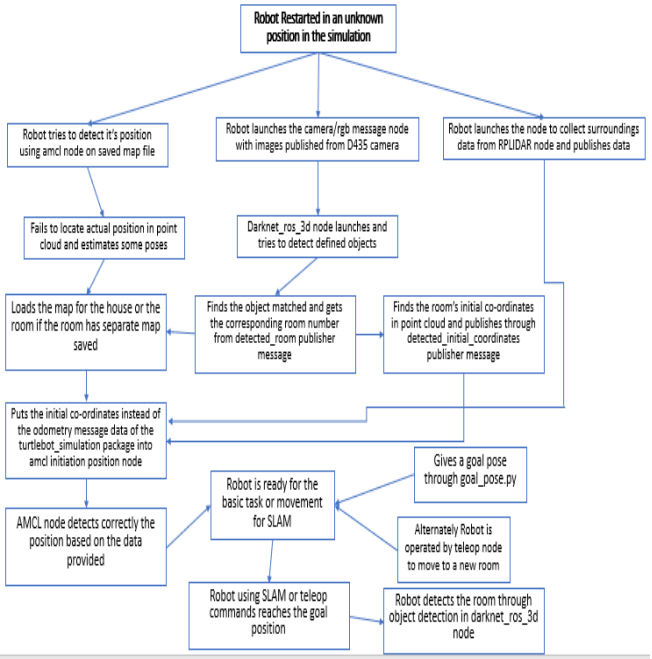

Figure 6: Pseudo Algorithm of the robot (p.s. zoom for better view)

As with the new update of gazebo PR2 robot's teleop node is not working properly we have used turtlebot3 burger model for teleoperation. But for better object detection PR2 robot simulation is used as the camera height of PR2 robot is better adjacent to the objects and turtlebot3 robot has ground visioning problem.

When the robot is restarted in an unknown position first it starts its RPLIDAR and Intel Realsense D435 camera node and the AMCL node to detect the position on the map. For real time practical environment AMCL should be started later after the object detection as the map needed to be confirmed first. But because of the robustness of the simulation the whole house map is pre-selected. The RPLIDAR constantly gives the room environment data to the AMCL package. When the camera detects the object (is real time robot may have to move around to find the object) as the object is visible at first glance of the robot in PR2 empty world, robot finds the corresponding room number analyzing the BoundingBox message given from darknet_ros_3d node and finding the class name of detected object.

```
room_dictionary = {“”diningtable”: “room1”, “person”:”room2”, “sofa”:”room3”, “”microwave”:”room4”, “suitcase”:”room5”, “hydrant”:”room6”}
```

```
room_initial_coordinates = {
‘room1’:{‘posx’:10.0, ‘posy’:20.0, ‘orix’:5.0, ‘oriy’:6.0, ‘oriz’:15.0, ‘oriw’:8.0}
}
```

Table 1: From where room and initial coordinates coming

Now at the autonav package based on turtlebot3 simulation when the PoseWithCovarianceStamped message is replaced by this initial poses and published at the init_pose.py files variable with also the map id in the message's header.frame_id then amcl automatically calculates standing in the position what is the orientation of the map and surrounding matched from RPLIDAR data.

```
init_msg.header.frame_id = “map”
init_msg.pose.pose.position.x = odom_msg.pose.pose.position.x
init_msg.pose.pose.position.y = odom_msg.pose.pose.position.y
init_msg.pose.pose.orientation.x = odom_msg.pose.pose.orientation.x
init_msg.pose.pose.orientation.y = odom_msg.pose.pose.orientation.y
init_msg.pose.pose.orientation.z = odom_msg.pose.pose.orientation.z
init_msg.pose.pose.orientation.w = odom_msg.pose.pose.orientation.w
```

Table 2: Where room and initial coordinates are put for AMCL package reading

When the goal pose is given to the robot through goal_pose.py node the AMCL package gets the ultimate map position of the robot after some movement. Here better SLAM algorithm can help the system to work with more accuracy. If the SLAM is not used then teleop node can be initiated and with keyboard “wasd” keys robot can be moved to different rooms to detect which room it belongs to. The publisher message that gives the room number is “detected_room” and initial co-ordinates giving publisher is “detected_initial_coordinates”. Both of them can be found in “room_detection_pkg” package's “detector.py” node. The code is available in github repository and some srdf files of the models in “pr2_tc_gazebo” package is more than 100mb and uploaded to the google drive.

IV. Result

As ROS is not in it's stable version still, it is very difficult to compile with the environment setup for ROS. The first problem facing with this experiment was installing nvidia drivers associated with tensorflow as YOLO algorithm uses tensorflow for the CNN building up for object detection. YOLO is fast incase of object detection because it uses Nvidia GPU along with CPU computational power. First it was planned to complete the project with raspberry pi. But as raspberry pi is a very small sized processor it can not handle YOLO computational power. Actual robots for object detection system with YOLO is built on Nvidia Jetson board. As ubuntu has the newest version of 22.04 (Jammy Jellyfish) all nvidia drivers and processor drivers are being updated to support the new OS version. In this case ROS is backdated and can support ubuntu 20.04 (Focal Fossa) for most of the simulation tools and packages being built and updated till this version. But the Nvidia drivers Cuda and Cudnn has updated to 12.xx series. Tensorflow in it's website has a real problem

tracking the compatible versions of nvidia drivers to support tensor built. In raspberry pi tensorflow can be installed and worked on anaconda environment. In that case YOLO will work with more computational time as running only on CPU power. But if YOLO has to be worked out with ROS then anaconda environment is not suitable because anaconda environment is built on Bazel system and ROS is built on Catkin system. The last solution to build the whole system environment after a lot of research is tensorflow updated version 2.4.0 with cuda 11.80 and cudnn 8.80 all installed separately. If one of the installation is failed the whole system gets corrupted and needs to rebuilt.

When the experiment is documented at the last days of its phase the gazebo and rviz got updated and suddenly pr2 teleop is not working properly. But turtlebot simulation teleop works properly so it was decided to detect object in the empty world and throw the initial value to read in the AMCL package associated with turtlebot simulation. But as per project initiation RPLIDAR and Intelrealsense D435 depth camera was present in experimental premises. So sometimes to test the real object is used with depth camera to be detected with darknet_ros node YOLO algorithm.

As lack of knowledge on srdf model making some of the object in the simulation was not fixed for example the umbrella. For this reason in some tests while teleop with keyboard, room containing microwave and umbrella gave wrong answers.

First we tested all the object in empty world. The things that were successfully detected later chosen for the object of that room as landmark.

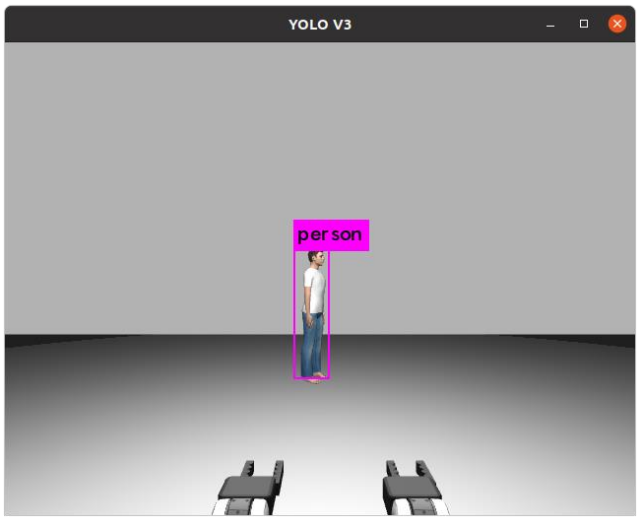

Figure 7: Checking person can be detected by YOLO

Then we also have tested with goal navigation automated with code. The result after 107 runs in simulation of gazebo is 91 times all the objects were detected correctly. Which means the accuracy is 85.047%. When the actual object detection was carried out, in 202 runs 188 times all objects were correctly detected. That concludes the accuracy of physical detection is 93.07%.

Here is the accuracy result table for simulation object detection:

Object to be detected	Number of correct detections	Number of wrong detections	Accuracy %
Umbrella	95	12	88.79
Person	107	0	100
Sofa	106	1	99.07
Microwave	98	9	91.59
Suitcase	93	14	86.92
Fire hydrant	105	2	98.13

Here is the accuracy result table for original object detection:

Object to be detected	Number of correct detections	Number of wrong detections	Accuracy %
Umbrella	200	2	99.0
Person	195	7	96.53
Sofa	192	1	99.07
Microwave	199	3	98.51
Suitcase	200	2	99.0
Fire hydrant (photo)	191	11	94.55

When the navigation goal is given to the robot because of not using an optimized SLAM algorithm only prebuilt turtlebot3 12 out of 52 runs the robot went to wrong room. In rviz gmapping SLAM algorithm is used to map the house first. So gmapping algorithm is used for SLAM in the whole project

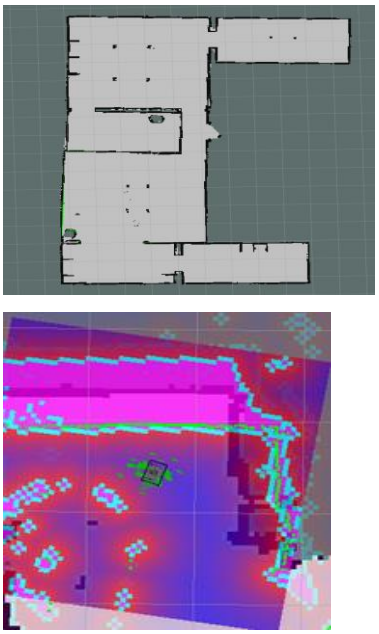

Figure 8: Rviz visualization of the amcl selection and the map of the house simulation

.

So incase of auto navigation the accuracy of the simulation is 76.92%.

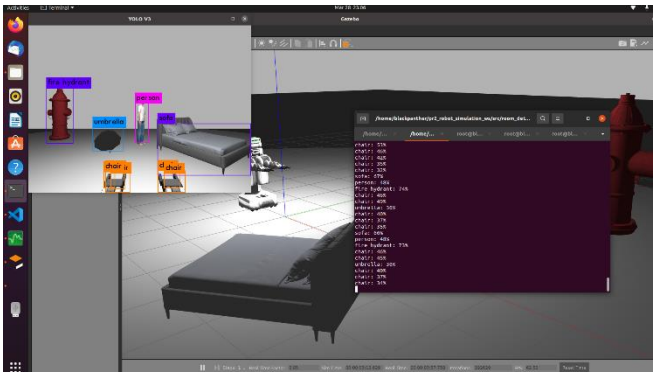

Figure 9: Checking of all objects that can be detected in simulation.

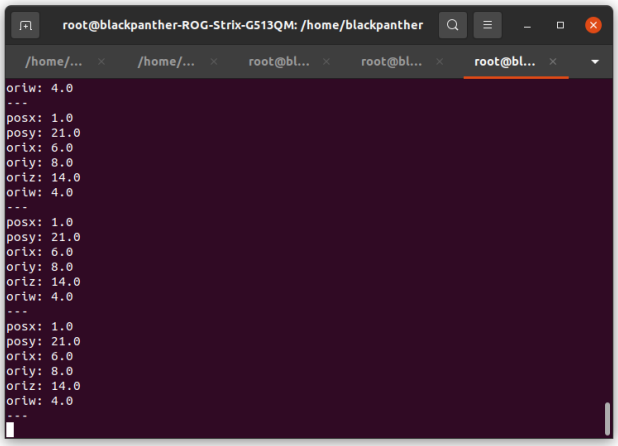

Figure 10: Checking Initial Position messages given from detected_initial_coordinates

As some of the srdf models were not complying being stable on the ground of the .world file simulation the simulation accuracy of object detection is lower than the real life object detection. Also the position given for the initial coordinates were wrong because of wrong object detection in simulation.

V. Future Scope

As ROS is a new system to be learnt by members enough time has to be spent on learning from two courses in udemy. For that reason, experiment is conducted only on simulation environment to check YOLO and AMCL algorithm's feasibility. For future scope it would be best to apply this algorithm with real turtlebot3 burger model in practical environment. The teleop of the PR2 robot got deprecated and needs to be fixed for newer version of gazebo simulation. The trajectory calculation and srdf models fitting requires trial and error method with knowledge of srdf model making to create stable object models in gazebo world. In the experimental system umbrella is not stable because of collision mechanism of trajectory calculation error. In accordance with this the object recognition system built on darknet_ros_3d node is not working with 100% accuracy even though the actual object in real depth camera is higher than the simulated once.

In the experiment firehydrant picture is used incase of testing the accuracy of YOLO object detection with real object. Firehydrant on streets are not easy to find now a days. To mitigate this error new objects should be trained for different rooms. To work with new training some parameters of darknet_ros package can be changed to work easily with real object and it is feasible to train for the actual environment before test run the real robot. For example, in rewe store shelves with lays chips can be detected as land marks for the kidnapped robot. Only lays picture should be pre trained in the YOLO object detection system of darknet_ros package.

Finally, the initial positions of the rooms are set manually in the message without calculation from the depth camera message for passing to the odometry messages. But setting initial pose messages should be done in automated way. In the future, it is possible upon the calculation through the sensor fusion of data acquired in 3D depth camera and LiDAR in point cloud for better mapping and distance calculation.

VI. References

Peng, G., Zheng, W., Lu, Z., & Liao, J. (2018, December). An Improved AMCL Algorithm Based on Laser Scanning Match in a Complex and Unstructured Environment.

Periwal, S. (2020, September 17). *Real-Time Object Detection with YOLO.* Retrieved from Latent View: https://www.latentview.com/blog/real-time-object-detection-with-yolo/

Saha, S. (2018, December 15). *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way.* Retrieved from Towards Data Science: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

SLAMTEC. (2021). *RPLIDAR 360 Degree Laser Range Scanner.* SLAMTEC.