Object Detection for Mobile Robot Navigation in Dynamic Indoor Environment

By: Anteneh Tilaye

A Thesis Submitted to the Department of Computing

School of Electrical Engineering and Computing

Presented in Partial Fulfillment for the Degree of Masters of Science in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

September 2019

Adama, Ethiopia

Object Detection for Mobile Robot Navigation in Dynamic Indoor Environment

By: Anteneh Tilaye

Name of Advisor: Prof. Yun Koo Chung

A Thesis Submitted to the Department of Computing

School of Electrical Engineering and Computing

Presented in Partial Fulfillment for the Degree of Masters of Science in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

September 2019

Adama, Ethiopia

Declaration

I hereby declare that this MSc thesis is my original work and has not been presented as a partial degree requirement for a degree in any other university, and that all sources of materials used for the thesis have been fully acknowledged.

Name: Anteneh Tilaye

Signature: _____

This thesis has been submitted for examination with my approval as a thesis advisor.

Name: Yun Koo Chung (PHD)

Signature: _____

Date of Submission: _____

APPROVAL OF THE BOARD OF EXAMINERS

We, the undersigned, members of the Board of Examiners of the final open defense by **Anteneh Tilaye** have read and evaluated his thesis entitled "**Object Detection for Mobile robot Navigation in Dynamic Indoor Environment**" and examined the candidate. This is, therefore, to certify that the thesis has been accepted in partial fulfillment of the requirement of the degree of Masters in Computer Science and Engineering (CSE).

Yun Koo Chung (PHD)	_____	_____
(Advisor)	Signature	Date
_____	_____	_____
(Chairperson)	Signature	Date
_____	_____	_____
(Internal Examiner)	Signature	Date
_____	_____	_____
(External Examiner)	Signature	Date

Acknowledgment

I would like to thank the almighty God for providing me everything I need in my life. Next, I would like to thank mother saint virgin Merry for the support I needed. I am also thankful for everything that crossed my path and the decision I made.

I would like to express the deepest appreciation to my advisor Prof. Yun Koo Chung for the encouragement and support by giving me brilliant comments and suggestions. He also invested in the research equipment I required for the completion of this research. He provided me the motivation to work in the robotics and deep learning area.

I sincerely thank my family Aynalem, Tilaye, Helen, Kelela, and Melat for providing me comfort and for being there when I needed them. I am also thankful for all my friends and my fellow Computer vision and robotics (CVR) SIG lab mates for the encouragement to finish my work.

Table of Contents

List of Figures

List of Tables

List of Equations

List of Abbreviations and Acronyms

BING	Binarized Normed Gradient
BOW	Bag Of Words
BRIEF	Binary Robust Independent Elementary Features
CNN	Convolutional Neural Network
CPU	Central Processing Units
EFK	Extended Kalman Filter
FAST	Features From Accelerated Segment Test
FLANN	Fast Library For Approximate Nearest Neighbors
GPU	Graphical Processing Unit
HOG	Histograms Of Oriented Gradient
HSV	Hue Saturation Value
LDS	Laser Distance Sensor
LIDAR	Light Detection And Ranging
MAP	Mean Average Precision
ORB	Oriented FAST And Rotated BRIEF
PCA	Principal Component Analysis
RANSAC	Random Sample Consensus
R-CNN	Region Convolutional Neural Network
ROS	Robot Operating System
SIFT	Scale-Invariant Feature Transform
SLAM	Simultaneous Localization And Mapping
SSD	Single Shot Detector
SURF	Speeded Up Robust Features
SVM	Support Vector Machine
TPU	Tensor Processing Unit
YOLO	You Only Look Once

Abstract

The detection and recognition of objects is a very important and challenging task in computer vision, as there is an increasing interest in building an autonomous mobile system. To make a mobile service robot truly ubiquitous to complex and dynamic indoor environments, they should be able to understand the surrounding environment beyond the capability of avoiding obstacles, navigating autonomously, and building maps. It's necessary to build a reliable and fast detection system to enhance the performance of indoor robot navigation. Several researchers have proposed different techniques for recognizing and localizing indoor objects for mobile robot navigation, however, most have low recognition rate, and some doesn't recognize multiple objects or does not run in real-time. Mobile robots do not come with heavy computing power so the detection algorithm response time should be sufficient enough that the robots can make a decision fairly quickly. Multiple object recognition and localization for mobile robot navigation in a dynamic indoor environment are proposed in this study. The algorithm detects an object by combining deep features for recognition and local features for localization. For recognition of multiple objects, a CNN based Deep learning model that uses MobilenetV2 as a base network with a sigmoid classification layer is developed. The model generates a probability for each object in the image independently, which is used for localizing the recognized object in the image. ORB+SURF is used as a feature extractor to generate a bounding box for each object. ORB+SURF is compared with different feature detection and descriptor combination to evaluate its performance. The object detection algorithm is also integrated with ROS to evaluate the performance in a real-world scenario and in doing so the distance information is also extracted from the Laser distance sensor mounted on the top of Turtlebot. Different navigation scenarios are executed by combining the object detection algorithm and the distance information in the test environment. The proposed object detection algorithm for mobile robot navigation achieves better performance in terms of recognition rate and speed, and also it provides the distance of the object to the robot.

Keywords: Indoor Object Detection, Convolutional Neural Network, Deep Learning, Object Localization, Mobile Robot Navigation, Semantic Navigation.

CHAPTER ONE

1 Introduction

1.1 Background

The Rapid evolution of technology and the development of robotic applications have made possible to create autonomous robots to assist or replace humans in ordinary tasks in an everyday setting. For instance, service robots should coexist with humans in the same environment to provide support and assistance in housework and caring for elderly or disabled people. Service robots usually work in an environment that is unstructured, where they collaborate with humans to accomplish tasks, but industrial robots are usually fixed in an environment that is structured and has external safeguards to protect them [1].

These service robots should be able to

- Build an internal representation of the environment and localize itself
- Have a set of capabilities that allows them to understand, interact and navigate in real environment and
- Understand commands from humans through various methods.

Vision is the most important sense for navigating or moving and interacting with the environment safely. Robots must have the ability to process visual data in real-time to adapt to the change to the environment. So for a robot vision, the detection and recognition of an indoor object is one of the important research topics in computer vision. Object recognition has a wide range of applications in the industry for example in inspection, manipulation, and registration tasks. Changes in illumination, occlusion, scales, and positions impose many limitations on object recognition performance. To make mobile service robots truly ubiquitous to complex indoor environments, they should be able to understand the surrounding environment beyond the capability of avoiding obstacles, navigating autonomously, and building maps.

It’s necessary to build a reliable and fast classification system to enhance the performance of indoor robot navigation. Various object classification system has been proposed over the last decade. Mobile robots do not come with heavy computing power so the detection algorithm

response time should be sufficient enough that the robots can make a decision fairly quick [2]. The algorithm to be developed should work under these specifications. In the past decade, there have been great advances in this area, though this issue still remains one of the most challenging problems in computer vision when a real-life scenario is considered.

In the world, 285 million peoples are estimated to be visually impaired and 82% of the peoples greater than 50 years and above were blind [3], [4]. Several approaches have been proposed to overcome this problem by developing a visual substitution system that could help blind people to navigate around. Most of the algorithms have a low recognition rate, and some don't recognize multiple objects or do not run in real-time. An object detection algorithm for mobile robot navigation is proposed in this study. The developed algorithm and robot navigation system can be used as travel assistance, aid to navigation and recognizing objects around the environment.

1.2 Motivation

Object detection is an important capability for robots to navigate in the real-world environment. Indoor environments continuously change, so the robot should be able to simultaneously map the environment as they navigate. Since robots are there to assist or help humans, it should recognize objects around it and know where to find them. Service robots are required to understand the environment to navigate and interact with the environment. Even though there have been great advances in the past decade, this issue still remains one of the most challenging problems in computer vision when a real-life scenario is considered. The robot builds the map of the environment and navigate using that map, and avoiding obstacles and collisions. However, the robot should also have to know the objects in order to provide information to the person that requires its assistance. This study tries to solve this problem by extending the solutions presented in previous works.

1.3 Statement of the Problems

Robots begin to enter the real world environment for providing services to people besides their importance in the industry. Service robots should interpret the environment and make a decision on the spot. The visual information provides a lot of information about the objects around the environment. The meaning and relationship between objects can be used to make a decision and

navigate. The detection and recognition of indoor objects for robot navigation is an essential task.

Recent works on object classification and localization problem for indoor robot navigation have the following gaps.

- The system does not recognize multiple objects in the same frame. If there are multiple objects of interest the system will be confused in [1], [5], [6].
- There work is not applicable in robot indoor navigation, since the system doesn't run in real-time [7].
- The number of objects chosen for testing is small and the techniques are specifically designed for each object, which does not work for other indoor objects or the recognition rate would be much less [8], [9].
- Efficiency depends on how the robots move and on the position and the distance of the robot from the object [2].
- Not robust in case of rotation, obstacle [10].
- The recognition rate is low [1] - [10].
- Distance information extraction is not efficient in terms of computation and speed[1], [2], [8].

In order to overcome these problems, it is necessary to build a reliable and fast classification system to enhance the performance of indoor robot navigation.

In this study, an attempt will be made to answer the following research question.

- How can the robot recognize multiple objects in the image considering resource constraints?
- What is the effective feature extraction technique for a robot to localize objects in the image?
- How to extract distance information about the object efficiently?

1.4 The Objective of the Study

1.4.1 General Objective

The general objective of the study is to design and develop multiple object recognition and localization for mobile robot indoor navigation.

1.4.2 Specific Objective

The following specific objectives will be addressed to achieve the general objective.

- Reviewing related works to understand the area and the works that are done by others.
- Developing image dataset for training and testing.
- Modeling and training the classification algorithm.
- Testing the trained model with a test set to find the best parameters.
- Finding fast and robust feature extraction algorithm for localization.
- Evaluating the performance of the object detection algorithm.
- Integrating the object detection algorithm with robot navigation.
- Evaluating the performance of the object detection algorithm in robot navigation.

1.5 Research Methodology

The following methods and techniques are applied in order to meet the objectives of this study.

1.5.1 Data Collection

This study uses a machine learning approach to solve the problem, so data is an essential part of the study. Images of indoor (i.e. office, lab, home) objects are also collected for both training and testing stages using a camera. Those data are collected directly (so primary data is used) from the indoor environment for the purpose of the study.

1.5.2 Literature Review

This study uses a literature review to enhance the research. Recent related literature is reviewed to get an insight into current trends and methods to solve the problem at hand. Necessary documents and tools are also reviewed for the development of the prototype.

1.5.3 Evaluation

The result will be analyzed to describe the performance of the object detection algorithm on a test data set. The performance of the object detection algorithm integrated with the navigation system will be analyzed in real-world scenarios.

1.5.4 Implementation Tools

For the development of the algorithm and prototype, open-source libraries like OpenCV, Tensorflow and Keras API will be used. The implementation platform will be in ROS (Robot Operating System) open-source operating system and Turtlebot3 mobile robot, which supports simultaneous localization and mapping, SLAM.

1.6 Scope and Limitation

1.6.1 Scope

The scope of the proposed work within the given time and resource includes:-

- ✓ Detecting objects in an indoor environment (classifying and localizing multiple objects).
- ✓ Integrating object detection module or node with Navigation module for Navigation.
- ✓ Providing distance information about the objects

1.6.2 Limitations

This paper does not cover the following due to time and resource limitations.

- **Detecting objects that higher than the robot height** is not a part of this work since the available robot is TurtleBot3 and its height would not allow it to see the object.

1.7 Significance of the Study

The application of the proposed work is to allow a mobile robot to differentiate between the objects in a scene to obtain information that can be used to make a decision. The object recognition algorithm can be used to assign a meaning to the scene in the environment and use the extracted information for different applications. Some of the applications are Semantic Navigation, Scene Recognition, and Environment categorization.

Generally, the outcome of this research has a significant impact on monitoring, security, military, and rescue operations. Specifically, since the thesis only focuses on the indoor environment it can be applied to service robots. The distance information can be used for manipulation and some other guidance systems. Some of the applications where this could be applied.

- ✓ Assisting an elderly person or person with a disability.
- ✓ Assisting Blind person to navigate in indoor environment.
- ✓ Message or object delivery service.
- ✓ Home security and surveillance
- ✓ Robotized wheelchair
- ✓ Floor cleaning

1.8 Organization of the Thesis

The thesis is organized as follows:

Introduction (Chapter One) describes the background of the area and the motivation, the statement of the problem and the research question to be answered, the methodology to achieve the objective of the study, the scope and limitations, and the application of the result.

Literature Review (Chapter Two) gives an overview of computer vision, object recognition and localization, mobile robot navigation, the challenge in object recognition algorithms, different techniques of object localization, previous works on object detection for mobile robot navigation, and comparison of different approaches taken by related works.

The methodology of the study (Chapter Three) discusses different methods and techniques used to develop the solution and select the appropriate one. Data collection method, design tools, prototype development framework and platforms, and evaluation methods are also discussed

Proposed Solution (chapter Four) presents the object detection algorithm and the integration of the object detection algorithm with the ROS for navigation.

Implementation (chapter Five) discusses the working environment setup and the implementation of the object detection algorithm and prototype for mobile robot navigation. It also discusses the integration of the object detection algorithm with the robot.

Result, Evaluation, and Discussion (chapter Six) present the result of the object recognition model and the localization algorithm for object detection and the evaluation. The result is discussed by comparing it with other related works.

Conclusion and Future Work (Chapter Seven) summarizes the work and conclude the result. It also presents future work.

CHAPTER TWO

2 Literature Review

2.1 Introduction

Computer vision is a field of science that focuses on how machines or computers make an understanding from digital images and videos. Computer vision tries to automate the tasks that a human visual system does. Some of the areas that computer vision can be used include agriculture, security and surveillance, transport, remote sensing, object recognition, medical image processing, process control, autonomous system. Currently, the development of an autonomous system is growing. For instance, self-driving cars and autonomous robots are one of the hottest research areas right now.

Industrial robots work in a clearly structured environment with external safeguards, but service robots usually work in an unstructured environment and collaborate directly with humans [1]. There are successful commercial service robots currently available on the market like Roomba and Scooba [11], for simple vacuuming or sweeping the floor. We can evolve these mobile robots by adding more complex tasks. To make a mobile service robot truly ubiquitous to complex indoor environments, they should be able to understand the surrounding environment beyond the ability to avoid obstacles, autonomously navigate, or build maps.

Vision is the most important sense for moving and interacting with the environment. For a robot vision, the detection and recognition of objects are one of the important research topics in computer vision. Object recognition has a wide range of applications in the industry for example in inspection, manipulation, and registration tasks. Object recognition and localization are important for robots to understand the environment, so it's necessary to build a reliable and fast object detection system to enhance the performance of indoor robot navigation.

An Object recognition algorithm takes an image as an input and returns a class label or category of the image and the class probabilities of objects that are present in that image as output. Object recognition algorithms have many limitations because of changes in illumination, scale, and other factors. Major challenges in object recognition include the following.

- **Illumination**: The light changes throughout the day, which affects the image of an object. Weather conditions and shadows also can affect the image. The same object in

different illumination may look different, making it difficult for the algorithm to discriminate.

- **Scale**: An object may appear in different sizes on different images. The selected feature should be robust enough to handle this change.
- **Rotation**: object recognition algorithm should handle rotation, as an object can appear rotated in the image.
- **Occlusion**: This is when an object is not completely visible or some part of an object is hidden. The algorithm should handle this condition.
- **Position**: the position of the object should not affect the success of the algorithm in recognizing the object.

Object detection algorithm outputs bounding boxes to indicate the location of the object in the image, in addition to the class label. So object detection is multiple object recognition and localization in an image.

2.2 Object Detection

An object recognition algorithm identifies which objects are present in an image, on the other hand, an object detection algorithm not only tells you which objects are present in the image but also outputs bounding boxes (x, y, width, height) to indicate the location of the objects inside the image. Different approaches have been proposed over the past decades for detecting objects in an image or video. Those techniques can be classified into two based on the kind of features used. The first class is based on a local feature like SIFT, SURF, etc. this class also includes the techniques that fuse these local features with 3D or Depth images. The second class is based on deep features, using deep neural networks for detecting objects in the images. This class can further be classified into two based on whether these algorithms use one or two stages for detecting the object. Those techniques are discussed in brief detail on the following subsections.

2.2.1 Object Detection Based On Local Features

In Local feature-based object detection, one or more features are extracted from the image and the object is modeled by using these features. The object of interest is represented by the shape, size or color of the object. This class can be further classified based on the type of local feature used.

2.2.1.1 Color-Based Object Detection

An object of interest is represented by its color information. Using color information for detecting objects in an image is not always appropriate. The feature set can be built using color histograms, gradient orientation histogram, edge orientation histogram, the properties of HSV color space and SIFT or SURF descriptors.

Zhenjun Han et al. [12] proposed an object tracking algorithm by combining color histogram (HC) bins and gradient orientation histogram (HOG) bins which consider the color and contour representation of an object respectively. S. Saravanakumar et al. [13] represented the object of interest by the properties of HSV color space. An adaptive k-means clustering algorithm was used to get color values centroid of the object

The most popular feature detector and descriptors are FAST, BRIEF, ORB, SURF, SIFT, and others for object detection and tracking. FAST (Features from Accelerated Segment Test) [14] is a computational efficient feature detector, which selects interest point by considering the intensity of 16 pixels circled around the pixel under test. If 8 pixels around the pixel under test are darker or lighter than that pixel, then it's selected as a key-point or interest point. FAST does not include orientation and multiscale feature. BRIEF (Binary Robust Independent Elementary Features) [15] is a fast feature descriptor which outperforms SIFT and SURF feature descriptor in terms of speed and recognition rate in many cases. BRIEF takes S by S patch around the key-point and creates a binary vector of size n (could be 128, 256 and 512). In the binary feature vector, 0 or 1 is set depending on whether the intensity of X is greater than the intensity of Y on a randomly selected pair (x, y).

ORB (Oriented FAST and Rotated BRIEF) [16] is an alternative to SIFT and SURF since SURF and SIFT are patented one should pay to use them for commercial use. It uses FAST feature detector with BRIEF feature descriptor. ORB modifies FAST feature detector to overcome orientation and multiscale problems. ORB is scaled invariant using a multiscale image pyramid to detect a key-point at a different scale. ORB uses Rotation-aware BRIEF to make it rotation invariant. It does this by steering BRIEF according to the orientation of the FAST key points. Another most popular algorithms are SIFT (Scale Invariant Feature Transform) [17] and SURF (Speeded up Robust Features) [18].

SIFT has four steps to detect and describe key points in the image. It finds potential key-point using the Difference of Gaussian by getting the Gaussian blurred image at different scale and finding the local extrema over scale and space by comparing each pixel with its 8 neighbors and 9 pixels in the next and previous scale. This local extremum is further refined by applying threshold value and 2x2 Hessian matrix to remove the edges [17]. Each key point is assigned an orientation to make it rotation invariant by making an orientation histogram with 36 bins covering 360 degrees. Lastly, 128 bin key point descriptor for 16 sub-blocks of 4x4 size having 8 bin orientation histogram is created [17].

SURF is created to speed up SIFT. SURF [18] uses Box filter instead of a Difference of Gaussian to approximate the Laplacian of Gaussian. Since convolution can be calculated with the help of an integral image, box filters can be convolved in parallel at a different scale. It uses the horizontal and vertical direction wavelet responses with Gaussian weight in the neighborhood of size 6 to assign orientation. Again for key-point description, SURF uses Wavelet responses in a horizontal and vertical direction with a neighborhood of size 20x20 around the key-point. SUFT feature descriptor has two versions, one with 64 dimensions and the other with 128 dimensions.

2.2.1.2 Shape-Based Object Detection

In Shape-based object detection, the object is represented by its shape. Usually done by extracting the contour of the object from the image. For extracting the contour RGB image or Depth images might be used. Huabo Sun et al. [19] proposed an object detection algorithm which detects edges with Canny method and extracts the contours of the object at different image resolution. L. Lu et al. [20] presented the representation of objects by HOG and PCA. The object is first transformed to the grids of Histograms of Oriented Gradient (HOG) descriptor and then apply Principal Component Analysis (PCA).

2.2.1.3 Template-Based Object Detection

Template-based object detection is usually done by matching the features between the template image of the object of interest and the image from the scene. This technique requires a template describing the object. This template can be fixed or deformable. It is a fixed template when the object shape does not change from a different viewing angle of the camera. For fixed template

matching can be done using image subtraction or correlation between the image from the scene and the template. Correlation is immune to noise and illumination effects in the images, whereas image subtraction should be done in a restricted environment. Template matching on deformable objects can be performed by applying parameterized deformation transform on the prototype (also called prototype-based template) [21].

2.2.1.4 Motion-Based Object Detection

The most common motion-based object detection techniques are thresholding technique over the inter-frame difference, Optical Flow and Gaussian Mixture. Gaussian Mixture models each value of a pixel by using a mixture of Gaussian for background/foreground segmentation.

2.2.2 Object Detection Based On Deep Features

Object detection ideas begin by searching region on the image and performing classification on detected regions. Over the recent years, most of the state of art object detections are based on one-shot detection, where the detect objects in the image through one pass. Generally, object detection based on deep features can be classified into two by whether these algorithms use object proposal or perform one-shot detection.

2.2.2.1 Object Detection Based On Object Proposal

Sub-regions (patches) of the image is selected first and then apply the object recognition algorithm to these image patches to detect objects. The location of the objects is given by the location of the image patches where the class probability returned by the object recognition algorithm is high. The straightforward approach to select patches is a sliding window, where we Crop multiple images by sliding window and pass each cropped image through ConvNet, but it's computationally very expensive.

This problem is solved by Object proposal algorithms. There are several object proposal algorithms like objectness measure [22], selective search [23] (used in R-CNN and Fast R-CNN), Binarized Normed Gradients (BING) [24], etc. Selective Search is one of the most popular region proposal algorithm used in object detection. It is based on computing hierarchical grouping of similar regions based on color, texture, size, and shape compatibility [23]. Selective

Search starts by over segmenting the image based on the intensity of the pixels using a graph-based segmentation method by Felzenszwalb and Huttenlocher [25]. Selective Search algorithm takes these over-segments as initial input and performs the following steps.

i. Add all bounding boxes corresponding to segmented parts to the list of regional proposals.
ii. Group adjacent segments based on similarity.
iii. Go to step 1.

At each iteration, larger segments are formed and added to the list of region proposals. Among object proposal algorithms Selective search is designed to be fast with very high recall. Binarized Normed Gradients (BING) [24] is another object proposal technique by Cheng et al. which is based on the observation that generic objects with well-defined closed boundaries can be discriminated by looking at the norm of gradients and it's the fastest.

It resizes the image to 8 by 8 and uses the norm of gradient as a sample 64D feature to describe it for training a generic abjectness measure. This is further binarized for efficient objectness estimation since it only requires a few atomic operations (ADD, BITWISE SHIFT, etc.). It can run 300fps on singe CPU laptops yielding a 96.2% object detection rate with 1000 proposals. As the number of proposals and color space increases, the detection rate also increases (99.5%). It's 1000 times faster than most popular alternatives selective search [23], Category-Independent Object Proposal [26], objectness measure [22].

There is a state of art general object classification and localization deep learning models like **R-CNN** (Region-Based Convolutional Neural Network), **Fast R-CNN**, **Faster R-CNN**, which are based on object proposal for localizing the object. Region-Based Convolutional Neural Network (R-CNN) [27] perform region search on the image using selective search and classifies each proposed region to one of the classes. It starts with small regions and hierarchically merges them to form a bigger region according to a variety of color spaces and similarity metrics [27]. After performing a selective search the output will be region proposals (~2k) which could contain the objects of interest. Each proposed region will be fed into the CNN model by resizing it so that the patch (region) will match the input of the model. The CNN will extract a 4096-dimension

vector of features, which are fed into multiple SVM classifiers to classify these regions to one of the classes by producing a probability that it belongs to each class.

R-CNN was able to achieve a 62.4% mAP score for PASCAL VOC 2012 test dataset and a 31.4% mAP score over the 2013 ImageNet dataset. Linear regression is used to modify the coordinates of the bounding box. R. Girshick [28] introduced Fast R-CNN to reduce the time consumption that is imposed on the R-CNN. Fast R-CNN extracts the feature using CNN from the entire image instead of applying CNN on each proposed region. The object proposal algorithm is applied to the feature map produced by CNN. ROI pooling layer is used to resize the feature map to a valid region of interest (ROI) with fixed height and width as hyperparameters. Each ROI is fed into fully connected layers, followed by a Softmax classifier. Fast R-CNN also uses linear regression to modify the bounding box.

Fast R-CNN achieved a 70.0% mAP score over the 2007 PASCAL VOC test dataset, 68.8% for the 2010 PASCAL VOC test dataset and 68.4% for the 2012 PASCAL VOC test dataset. Faster R-CNN is proposed by Shaoqing Ren et al. [29], which uses Region Proposal Network (RPN) instead of a computationally expensive selective search. The entire image is fed into a pre-trained (on ImageNet) CNN model to extract the feature and RPN proposes a maximum of k regions. The classification and bounding box prediction on the proposed is done by using two fully-connected layers. Faster R-CNN is just the combination of Region Proposal Network and Fast R-CNN, in which Faster R-CNN replaces the selective search by RPN. Fast R-CNN obtained a 78.8% mAP score over the 2007 PASCAL VOC test dataset and 75.9% over the 2012 PASCAL VOC test dataset.

Faster R-CNN 34 times faster than the Fast R-CNN by using RPN. All the above models use object proposal algorithms like selective search and RPN (region proposal network) to localize objects. Although these models have high accuracy (achieved promising mean Average Precision (mAP)), they are computationally expensive, since they have to run classification for every proposed region.

2.2.2.2 One-Shot Object Detection

YOLO (You Only Look Once) [30] and SSD (Single Shot Detector) [31] predicts bounding boxes and class probabilities with a single network in a single evaluation instead of proposing objects and classifying each proposed window.

YOLO takes the entire image as input and divides it into S x S grid. Each cell in the grid predicts B bounding boxes with a confidence score. The confidence score is the probability that there is an object multiplied by the IOU (intersection over union) between the predicted and ground truth bounding box. The output of the final layer is a S x S x (C + B x 5) tensor corresponding to each cell the grid [30]. S represents the size of the grid cell and C is the number of estimated class probability. B is the number of anchor boxes, each having 4 coordinates and 1 confidence value. YOLO preforms non-maxima suppression at the end of the network to merge highly overlapping bounding boxes. YOLO has the total number of convolutional layers are 24 followed by 2 fully connected layers. It also have a less accurate and fast version with 9 convolutional layers and fewer filters [30].

YOLO achieved a 63.7% mAP score over the 2007 PASCAL VOC dataset and a 57.9% mAP score over the 2012 PASCAL VOC dataset. W. Liu et al. [31] have developed a Single Shot Detector (SSD) that is similar to YOLO. SSD also predicts the bounding boxes and class probability in a single shot with end to end CNN architecture. SSD uses feature maps at different positions of the network to predict the bounding boxes. The image will be passed through different convolutional layers which are having different sizes of filters. 10 x 10, 5 x 5, 3 x 3 filter sizes are used in SSD, whereas YOLO use 1 x 1 and 3 x 3 filters. To generate the bounding boxes SSD uses extra feature layers (convolutional layers with 3 x 3 filters) at different positions of the network [16]. They have obtained mAP scores of 83.2% over the 2007 PASCAL VOC test dataset and 82.2% over the 2012 PASCAL VOC test dataset [31].

J. Redmon et al. [32] introduced a second version of YOLO in order to increase accuracy while making it faster. Accuracy improvements are made by using batch normalization instead of dropouts, the high-resolution classifier (448 x 448 picture), convolution with Anchor Boxes, removing fully connected layers, Fine-Grained features where is used feature maps of different layers by reshaping them to the same dimension [32]. Speed improvements are achieved by

replacing VGG16 by customized GoogLeNet (requires less operation) and using DarkNet to further simplify the backbone CNN used [32]. YOLOv2 achieved 78.6% mAP on VOC 2017. YOLOv2 is replaced with a more accurate and faster version called YOLOv3. YOLOv3 [33] replaced softmax function with independent logistic classifiers to calculate the class probability and uses binary cross-entropy loss for each label. They also introduced Feature Pyramid Network (FPN) like Feature Pyramid. YOLOv3 makes predictions at 3 different scales. It processes images at 30 frames per second (FPS) and has an mAP of 57.9% on the COCO test-dev on Pascal Titan X.

These models achieved promising mean Average Precision (mAP) and they can run real-time on computationally demanding machines [32]. Mobile robots do not come with heavy computing power so the detection algorithm response time should be sufficient enough that the robots can make a decision fairly quick [2]. The algorithm to be developed should work under these specifications.

2.3 Object Detection for Mobile Robot Navigation

A mobile robot is a robot that is capable of moving around the environment and not fixed or stationed to one physical location. Mobile robots have the following functional characteristics.

- **Mobility**: it should have total mobility relative to the environment (land, air, water).
- **Perception ability**: it should have the ability to sense and react to the environment.
- **A certain level of autonomy**: there should be limited human interaction.

Mobile robots can be classified based on the environment it resides. There are land-based robots (legged, wheeled and tracked), water-based (boats, submarines) and air-based robots (helicopter, blimp). Usually, industrial robots are more or less stationed, having an arm or manipulator and attached to fixed surface while mobile robots should able to move from one place to another for accomplishing a task. One of the categories that can be taken as a mobile robot is service robots. Service robots are autonomous and mobile agents designed to assist or give service to humans in order to perform an everyday task in a domestic environment. Service robots are expected to coexist with a human in order to provide support or assistance in the work area and care elderly or disabled people.

Service robots work in an environment that is unstructured and it directly interacts with humans. The environment that human lives are meant for humans and service robots are expected to work in the same environment. Service robots need certain characteristics in order to provide these supports.

- It should be able to build an internal representation of the environment and localize itself in that environment.
- It should be able to navigate through the environment.
- It should be able to plan a path and what to do under different scenarios.
- It should understand and interact with the environment it resides in.
- It should interact and understand commands from humans through different means.

Perceiving the environment can be done with different sensors. The most important sensor to understand and navigate through the environment is a vision. The following subsections discuss more in a brief detail on mobile robot navigation for service robot navigation and localization and object detection for understanding the environment.

2.3.1 Mobile Robot Navigation

For mobile robots, the ability to navigate in the surrounding environment is important. The robot should avoid situations like collusion, unsafe conditions, but also should accomplish its purpose to navigate in the surrounding robot environment. Robot navigation is the ability of a robot to reach a desired location with the ability to position itself in the current environment and plan a path to the desired location. Mobile robot navigation is can be defined as a combination of three fundamental components.

i. **Self-localization** is the ability to establish or realize its own location and orientation within the frame of reference or coordinate.
ii. **Path planning**: is the realization of the current location and destination of the robot and planning how to navigate to the destination from the current location within the same frame of reference or coordinate. The robot should find the best route and navigate to the destination.
iii. **Map building and interpretation**: is the ability of the robot to build and interpret the notation or map describing locations in the robot frame of reference. A map is the

representation of the robot environment, in which the robot can refer to understand the environment layouts and locations.

Some mobile robot navigation systems have the ability to perform Simultaneous Localization and Mapping (SLAM).

2.3.2 Simultaneous Localization and Mapping (SLAM)

SLAM deals with the problem of mobile robot navigation or building a map of an unknown environment while at the same time navigating the environment using the map and localizing itself [34]. It's a process where a mobile robot builds its own map and realizes its current location simultaneously while navigating the unknown environment. The trajectory of the platform and the location of the landmarks are estimated online without any prior knowledge of the environment. SLAM consists of different parts like Landmark extraction, data association, state estimation, state and landmark update. SLAM has many different steps and these steps can be implemented using a number of different algorithms [34]. The outline of the SLAM process is given below.

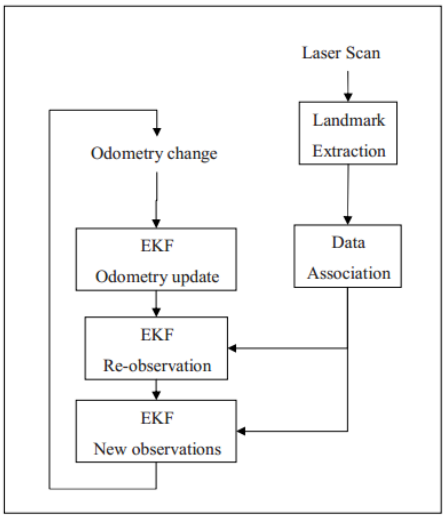


Figure 2.1 Outline of the SLAM Process

i. Odometry Data

Odometry data is an approximate position of the robot measured by the movement of the wheels of the robot. Odometry data is an initial guess of where the robot might be in. we cannot use the

odometry of the robot directly as it is often erroneous. To correct the position of the robot, laser scanners can be used to scan the environment.

ii. Landmark Extraction

We can use the laser scans of the robot environment to extract the features from the environment and re-observe when the robot moves around [34]. The feature extracted from the environment is called landmarks. These features are used to find out where the robot is or for a robot to localize itself.

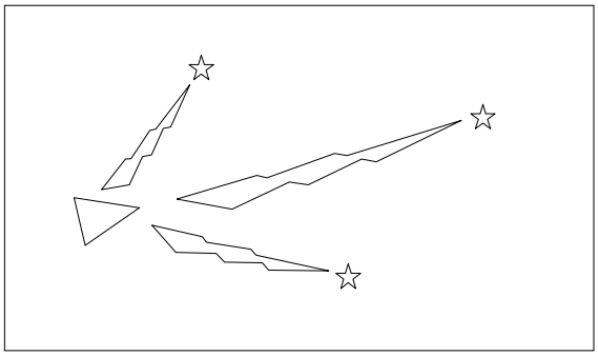


Figure 2.2 Landmark Extraction

In the above diagram, the triangle is a robot and stars represent landmarks. The lightning represents the sensor measurement, where the robot measures the location of the landmarks by using its sensors. Characteristics of landmarks include it should be easily re-observable from different positions and angles. Individual landmarks should be distinguishable or unique from each other. It should be plentiful in the environment so that the robot doesn't get lost. Landmarks should be stationary. There are multiple ways to do landmark extraction depending on the type of landmarks and sensors used. For example, spike landmarks are used for non-smooth surface and RANSAC is used for a smooth surface.

iii. Data Association or Re-Observing Landmarks

The extracted landmarks should be stored in a database, the landmark should be observed N times before it's stored. After performing a new laser scan and extract the landmarks, each extracted landmarks are associated with the closest landmarks in the database. The nearest-neighbor approach can be used to associate a landmark with the nearest landmark in the database.

iv. State Estimation and Landmark Update

State or position estimation is done by Extended Kalman Filter (EKF) from the odometry data and landmark observations [34]. An EKF is the heart of the SLAM process. The EKF keeps track of an estimate of the uncertainty in the position of the robot and also the uncertainty in these landmarks it has seen in the environment [34]. After the landmark extraction and the data association, the SLAM process can be considered as three steps:

- Update the current state estimate using the odometry data. When the robot moves to the new position it will be updated using odometry update.
- Update the estimated state from re-observing landmarks. The extracted landmarks will be associated with the observation of landmarks it previously has seen. The re-observed landmarks are used to update the position of the robot in the EFK.
- Add new landmarks to the current state. The landmarks which have not been seen are added as a new observation.

2.3.3 Related Works

Mobile robots do not have heavy computing power, so in order to make the robot move fairly quick, the response time to detect an object should be sufficiently efficient [2]. The object detection algorithm to be developed should work under these specifications. Various object classification and detection systems have been proposed over the last years. A most common approach is using a local feature of an RGB image to represent object features. H. Lee et al. [10] used structural features of pillar and hallway from RGB images with a decision tree for recognition and they did correlation with template mask to detect the pillar corner point. Takács et al. [1] Applied Speeded Up Robust Features (SURF) Feature Detector and Descriptor with Bag of Words (BOW) and Support Vector Machine (SVM) for recognition of indoor objects (such as chair, fire extinguishers, and trash can), and they did SURF Localization on the classification result to localize the object in the frame. H. Jabnoun et al. [5], [35] used SIFT Feature and SIFT localization for identifying daily life objects. N. Diriba [36] proposed a sign detection algorithm for robot navigation using ORB and SURF, where the feature is matched to detect signs in the scene image. W. Obaid et al. [6] took color histogram of the patch for detection and SIFT feature matching between the model and the patch.

Some researchers [2], [8], [9], [37] applied depth information and 3D information for the segmentation of the object from the scene. In [8] Feature is extracted from depth image and RGB image and fed to SVM for classification, the techniques are designed specifically for every 3 objects. Hernandez et al. [9] added uncertainty calculation on top of [8] work. In [23] Point Cloud or 3D data is used to segment a horizontal plane and detect the object placed on it and they explore 2D classification algorithms with SURF and Scale Invariant Feature Transform (SIFT) features. Astua et al. [2] used two methods, contour extraction, and FLANN Matching, to segment the object and the size of a contour and SURF features are used with correlation and FLANN to classify the Images.

Recently, deep learning also gained increasing attention in the computer vision area for recognition, detection and Image segmentation. X. Ding et al. [7] proposed a Convolutional Neural Network (CNN) architecture with a selective search for object proposal and Detection Fusion to refine indoor object recognition for indoor object recognition. Y. chang [38] also proposed a Faster R-CNN based algorithm for object detection for ROS based mobile robots using GPU-accelerated computing. The deep learning approaches presented above do not have real-time speed. In the past decade, there have been great advances in this area, even though this issue still remains one of the most challenging problems in computer vision when a real-life scenario is considered.

2.3.4 Summary of Related Works

The following tables illustrate previous works on object recognition and localization. The selected papers are from 2014 onwards.

Table 2.1 Previous Works on Object Recognition and Localization Part 1

	Object Recognition to Support Indoor robot navigation, 2015 M. Takács et al. [1]	**Object Classification in Natural Environments for Mobile Robot Navigation, 2016 Hernandez et al.** [8]	**Object Detection Techniques Applied on Mobile Robot semantic Navigation, 2014 C. Astua et al.** [2]

Dataset (Data collection)	- 244 Images, 8 classes taken by 640 x 480 resolution Kinect sensor.	- Depth with RGB images, 3 objects, taken by ASUS Xtion Pro Live	- Depth images, taken by Kinect sensor
Preprocessing or enhancement	- No preprocessing is used to enhance the image	- Equalization, morphological operations, Gaussian filter, and thresholding.	- equalization, morphological transformations
Segmentation or object proposal	- No segmentation techniques is applied. SURF localization is used.	- Contour extraction, Hough transform, and watershed. - A different technique for each object.	- Two methods - Contour extraction - FLANN Matching
Feature extraction	- SURF - Bag of visual Feature (BOW) is used to create a codebook	- Geometric features - Closet – solidity, extent, circularity, handle ratio. - Chair – circularity. - Screen – circularity, extent and aspect ratio.	- Size of a contour for the first method - Speeded Up Robust Feature (SURF) for the second method - Combination of both.
Classification	- Support Vector Machine (SVM)	- Support Vector Machine (SVM). - Two Approaches - One against all and one against one.	- Correlation for the first method - Fast Library for Approximate Nearest neighbor (FLANN) for 2nd
Evaluation	- 85 % accuracy	- 1st - (81.58%) closets, chairs (72.79%), screens (65.60%) - 2nd - 81.97%, 76.56%, 60.13%	- Not very computationally demanding
Limitations	- The system does not recognize multiple objects	- Low recognition rate. - A small number of objects.	- Efficiency depends on ✓ How the robot moves.

	in the same frame. - Low recognition rate.	- The techniques are not generalizable.	✓ Position and the distance of the robot from the object

Table 2.2 Previous Works on Object Recognition and Localization Part 2

	Object recognition for vision-based navigation in an indoor environment without image database, H. lee et al. 2014 [10]**.**	**Indoor Object Recognition Using Pre-trained Convolutional Neural Network, X. Ding et al. 2017** [7]**.**	**Visual substitution system for blind people based on SIFT description, H. Jabnoun et al. 2014** [5]**.**
Dataset (Data collection)	- 24 RGB images captured smartphone camera (640x360), 4 classes. - No image database	- Public indoor dataset (18 categories) and private FoV (17 categories).	- Private video frames, the number of classes are not specified.
Preprocessing or Enhancement	- Edge extraction	- Scaled (to 256x256). Since Caffenet expects with that size.	- All images are converted to grayscale.
Segmentation or Object Proposal	- Correlation with template mask to detect the pillar corner point.	- Selective search method is used to generate ROI with bounding boxes.	- No localization of an object.
Feature Extraction	- Structural features of the pillar, hallway and hallway entrance.	- CNN pipeline is used to extract features	- SIFT (Scale Invariant Feature Transform)

Classification	- Decision tree. - Comparing metrics of four features (pillar, hallway, hallway entrance)	- CNN by using Caffenet as a reference model.	- SIFT features of the target image are matched with the database
Evaluation	- From 24, 17 images are well recognized - Recognition rate - 70.8% - 50% pillar, 25% entrance, 83.3% hallway , 90% absence.	- Mean average precision (mAP) of 84.2%. - Detection fusion is used to reduce misclassification.	- The evaluation technique is not stated.
Limitations	- Low recognition rate - Not robust when rotation, obstacles	- The developed algorithm does not run in real-time.	- The system does not recognize multiple objects in the same frame.

Table 2.3 Previous Works on Object Recognition and Localization Part 3

	Object Detection and Identification for Blind People in Video Scene, H. Jabnoun, 2015 [35]**.**	**Real-Time color object recognition and navigation for QUARC QBOT2, W. Obaid et al. 2017**[6]**.**	**Adding Uncertainty to an Object Detection System for Mobile Robots, C. Hernandez et al. 2017** [9]**.**
Dataset (Data collection)	- Private Dataset, 4 videos sequences (daily life objects).	- Images captured by Microsoft Kinect.	- Depth with RGB images of 3 objects, taken by RGB-D Camera

Preprocessing or Enhancement	- This paper is different from the [5], in this they used color information	- Split the scene into patches of 30x30 pixels	- equalization, morphological operations, Gaussian filter and thresholding based on [8]
Segmentation or Object Proposal	- SIFT localization is used.	- Every patch is represented by Averaged histogram of RGB values of every pixel with its 8 neighbors - Apply Histogram intersection between every patch and determine the highest intersection	- Contour extraction, Hough transform, and watershed. - A different technique for each object based on [8]
Feature Extraction	- SIFT (Scale Invariant Feature Transform) from the color image.	- Color histogram for detection - SIFT features	- Geometric features - Closet – solidity, extent, circularity, handle ratio. - Chair – circularity. - Screen – circularity, extent and aspect ratio.
Classification	- SIFT features of the target image are matched with the database	- Performing SIFT between the model and the patch with the maximum intersection	- Support Vector Machine (SVM) - uncertainty calculation is added on top of [8]
Evaluation	- 95% true positive when the scale is 5 for the SIFT algorithm.	- 86% match in 1.2 seconds on windows 10 Intel core i5.	- The detection rate is the same as [8], only uncertainty is added
Limitations	- Detection failure caused by the quality of the image, the size of the	- Low recognition rate - Recognition and Localization of multiple	- Low recognition rate. - A small number of objects.

	target object (small), the high speed of the video scene.	objects in the same frame are considered.	- The techniques are not generalizable.

As shown in the above tables, simple features like color histogram, SIFT or SURF are used to represent object features and in some of the works Depth images along RGB image are used. Some applied deep features for indoor object recognition. Object proposal techniques (i.e. Selective search), SURF or SIFT localization and Contour Detection are used for localizing objects in the image. Even though these previous related works scored promising success, they suffer from problem or limitations like low recognition rate, a small number of objects, techniques are not generalizable, not robust when rotation and occlusion, not recognizing multiple objects at the same frame, slow speed (not real-time) and other problems. In order to overcome these problems, it is necessary to build a reliable and fast classification system to enhance the performance of indoor robot navigation.

This study tries to solve this problem by extending the solutions presented in previous works and using current and new trends.

CHAPTER THREE

3 The Methodology of the Study

3.1 Overview

In this chapter, the research methodology and the process to develop the object detection for mobile robot indoor navigation is explained. Available development tools will be compared in order to select the appropriate one for the proposed system.

In order to achieve the objective of this study, the following methods and techniques are employed.

3.2 Data Collection Method

This study uses a machine learning approach to solve the object detection problem for robot navigation, so data is an essential part of the study. Images of indoor (i.e. lab) objects are collected for both training and testing stages using a camera. Those data are collected directly (so primary data is used) from the indoor environment for the purpose of the study.

Since object detection is a computer vision problem the data is an image. The images are collected from the computer vision lab in my university and testbed created for this research. Firstly video of an object is grabbed using a camera and frames are extracted from the video to get the training and testing dataset. TECHNO L8 Lite is used to record the video, which is capable of recording HD with 720p and 1080p. The frame size in the video is 1920 by 1080 pixels, so the image is cropped to different smaller sizes depending on the type of the object, using Video Crop mobile application. The frames are resized to 256 by 256 pixels image after cropping them.

3.3 Feature Extraction Algorithm Selection

The proposed object detection algorithm has two steps to detect the objects in the image. Different feature extraction algorithm is proposed for both steps, deep feature based feature extractor for recognition or probability generator and local feature-based feature extractor for localization is used. The selection criteria include the speed, accuracy, and number of data. More detail is given in the following subsections.

3.3.1 Feature Extraction for Recognition

Several states of art Deep CNN based architectures have been proposed over the last decades for classification of images. These different state of art CNN based feature extraction architecture include ResNet, VGG, Inception, Xception, MobileNet, and others.

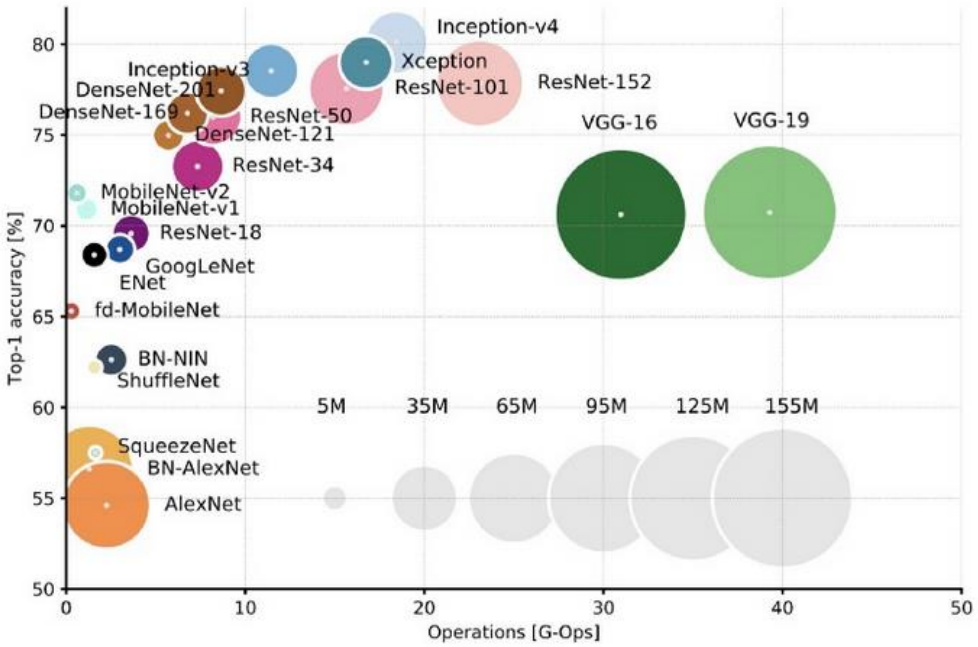


Figure 3.1 Benchmark Analysis of Deep Neural Network Architectures [31]

The above figure shows the top1 accuracy vs. the number of operations, parameters based on the work of S. Bianco et al [39]. The size of the blob represents the number of parameters. The x-axis represents the number of operations performed in a single forward pass. MobileNetV2 has less than 5 million parameters where others architectures like VGG has more than 155 million parameters. MobileNetV2 has less number of parameters and operations compared to most architectures, which makes it able to run fast on different devices that have less computing capabilities. There are other architectures that have less number of parameters and operations like fd-MobileNet as shown in the figure, but they are less accurate. Based on the above observation MobileNetV2 is used in this research for feature extraction in the task of generating the probability of each class from the given image.

3.3.2 Feature Extraction for Localization

There are different feature selection and description algorithms. The most popular feature selection algorithms are SIFT, SURF, ORB, FAST and BRISK, and descriptors are SIFT, SURF, BRISK, ORB, BRIEF. In order to detect an object in an image finding the best combination of feature detector and descriptor is an important task. The algorithm to be chosen should be fast enough for the robot since the robot needs to decide what to do after the object detection. For this research ORB-SURF combination of feature detector and descriptor is used based on the work done by P. Boyraz et al [40] and comparing them with different parameters. The authors [40] performed a detail performance comparison on different 23 combinations of feature detectors and descriptors that are listed above. The parameters used are speed, accuracy, and number of correct matches per second. From these combinations the highest accuracy is achieved by SURF feature descriptor is used and SIFT-SURF combination has achieved the highest accuracy but on my dataset ORB-SURF combination performs better.

3.4 Development Tools

For this research different types of development tools are used to design and implement the proposed system. The development tools section gives a description and justification of these development tools. These tools include prototype development tools and platforms, UML Modeling tools, and other tools that are relevant to the research. The following sections give a brief detail about these development tools.

3.5 Design Tools

Design tools are media that are used for the production, expression, and perception of design ideas. Edraw Max [41] is used to design in the proposed system. It is a lightweight and powerful graphic design tool for creating professional-looking flowcharts, network diagrams, UML diagrams, and others. This tool is selected because [41].

- It has lots of high-quality shapes, example, and template,
- It easily visualizes complex information with a wide range of diagrams.
- Works with MS Office well and others.

3.6 Prototype Development Frameworks

3.6.1 Tensorflow

Tensorflow is an open-source software library, which is developed for high-performance numerical calculation and computation. Its flexible architecture that can be deployed easily on a variety of platforms like Central Processing Units (CPUs), Graphical Processing Units (GPUs), Tensor Processing Unit (TPUs). It also can be deployed on desktops computers, clusters, mobile, and edge devices. Google Brain team (a group of researchers and engineers) is the one that developed this library. It supports machine learning, deep learning, and flexible numerical computation [42]. The following diagram shows the power score of deep learning framework based on usage, interest, and popularity [43].

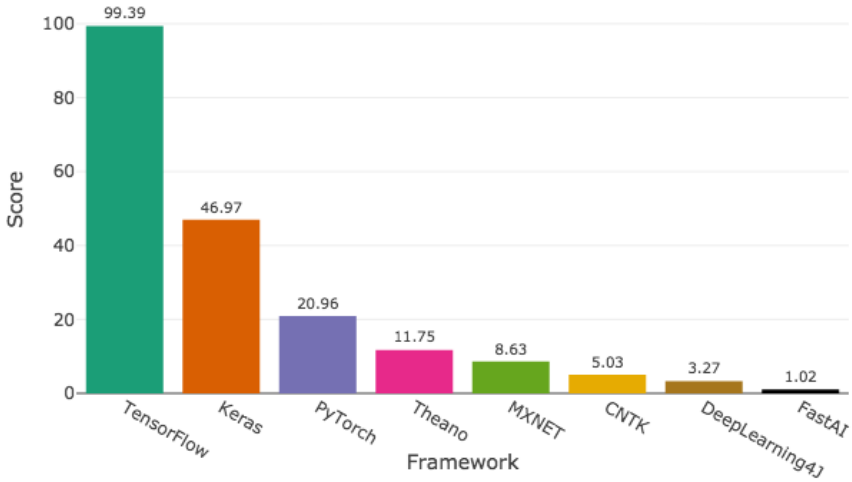


Figure 3.2 Deep Learning Framework Power Score 2018 [44].

As shown in the above diagram, Tensorflow is by far the most used and popular deep learning framework.

3.6.2 Keras

Keras is a high-level API built on Tensorflow (and can be used on top of Theano too). Compared to Tensorflow it is more user-friendly and easy to use. Keras is user-friendly, supports Modularity, Easy extensibility and works with Python [45]. Keras has the following properties.

- It is user-friendly, modular, and extensible

- It allows easy and fast prototyping
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- It can run runs seamlessly on CPU and GPU.

Keras is the 2nd most popular and used deep learning framework according to the above diagram.

3.6.3 OpenCV

OpenCV is an open-source computer vision and machine learning software library built to provide a common infrastructure for computer vision applications [46]. It has C++, Python, Java and Matlab interfaces and also supports Windows, Linux, Android, and Mac OS. OpenCV was made for image processing, so each function and data structure was designed with the image processing engineers in mind.

3.7 Prototype Development Platforms

3.7.1 ROS

ROS is an open-source, meta-operating system intended to be used with a robot. It provides the services that are expected from an operating system, like hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management [47]. It provides a collection of tools, libraries, and conventions with the purpose that developing complex and robust robot behavior can be simplified in different robotic platforms [47].

Several styles of communication are supported and implemented by ROS. Some of the communication includes synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server. An executable file within a ROS package is called Node. The ROS client library is used by ROS nodes to communicate with each other. If one node needs to communicate with other nodes it should publish or subscribe to a topic. Message is a ROS data type which holds the information that node is sending each other. The goals of ROS can be summarized as:

- Peer-to-peer

- Tools-based
- Multi-lingual
- Free and Open-Source

✓ **Simultaneous Localization And Mapping (SLAM)**

SLAM is the computational problem of building or updating a map of an unfamiliar environment while at the same time keeping track of the robot location within it. ROS have a gmapping package, which provides laser-based SLAM (Simultaneous Localization and Mapping), as a ROS node called slam_gmapping. Using slam_gmapping, a 2D occupancy grid map can be created (like a building floorplan) from a laser and pose data collected by a mobile robot [47].

✓ **3D Visualization Tool**

The 3D visualization tool of ROS is Called RViz. It is designed to show ROS messages in 3D so that we can visually verify data. It can visualize data received from various sensors such as LDS, RealSense and Kinect or camera. The Robot is described with the Unified Robot Description Format (URDF) in ROS. URDF can be expressed as a 3D model that can be used for simulation and control [47].

✓ **ROS GUI Development Tool**

RQT is a software framework provided by ROS that implements the various GUI tools in the form of a plugin. It can be used to show the hierarchy of each node as a diagram to show the status of the current node and topic, plot the messages as a 2D graph.

RQT contains three meta-packages [48]:

- RQT: core module
- RQT common plugins: ROS backend tools.
- RQT robot plugins: tools that are used to interact with robots during their runtime

3.7.2 Turtlebot3

Turtlebot is a ROS standard platform robot. It has three versions, the first is Turtlebot1, developed by Tully and Melonee from Willow Garage on top of the iRobot's Roomba-based research robot in 2010. In 2012 Turtlebot3 was developed Yujin Robot based on the iClebo Kobuki research robot. Turtlebot3 was developed to overcome the lack of function by its predecessors.

TurtleBot3 is a new generation mobile robot that's modular, compact and customizable for education, research and product development [49]. It supports ROS and its features like extensibility, modular, strong sensor lineups (highly utilized raspberry pi camera, Enhanced 360^0 LIDAR) and open-source hardware and software, makes it a good choice.

The TurtleBot3's core technologies include Simultaneous localization and mapping (SLAM), Navigation and Manipulation (using something like Open Manipulator), making it suitable for home service robots [50].

3.8 Evaluation Method

The result will be analyzed to describe the performance of multiple object recognition model on a test data set. The dataset is split into different training and testing set using different test sizes. The algorithm is evaluated using the test set. A confusion matrix is used for evaluating the performance of the model. From the confusion matrix the model's accuracy, mean Average Precision (mAP), recall and F1 score is calculated.

CHAPTER FOUR

4 Proposed Work

4.1 Overview

This chapter presents the proposed solution to the object detection problem for mobile robot indoor navigation. The service robots should be able to represent and understand the environment and should be able to communicate with humans to provide support. In order to make this possible object detection, semantic navigation and human-robot interaction are important. This chapter is divided into two sections. The first section presents the proposed object detection algorithm in order to understand the environment. The second section explains how the internal representation of the environment is constructed and how the robot navigation is integrated with the object detection algorithm.

4.2 Object Detection

For a service robot to understand the environment, detecting objects or knowing different kinds of objects in the scene is important. An object detection algorithm should be used to interpret the environments by finding the objects around the robot. The proposed object detection algorithm is a combination of two steps. The first one is recognition, which is responsible for providing the probability of each class for a given image. The output of the recognition step is used to localize the object in the image, so the second step is responsible for localizing the object based on the probability provided by the recognition step.

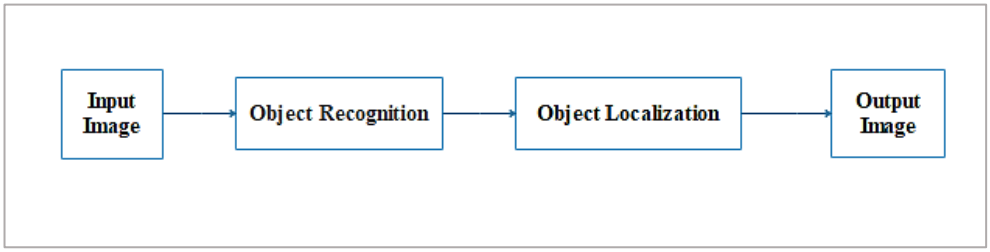


Figure 4.1 Object Detection Overview

The detection of objects around the environment should be fast enough so that the robot makes some decisions based on the detection output. The proposed algorithm should consider this constraint. The reason why object recognition is need before localization using matching, the algorithm only has to localize once it is sure that the object exists in the image. If the object

recognition is not there, the algorithm has to match every database image with a scene image. This approach is not efficient because if there are thousands of objects in the database, it takes a lot of time to detect an object for one image. In the proposed algorithm the recognition step is responsible to generate the existence probability for each object of interest. The following subsection describes the proposed algorithms with more detail.

4.2.1 Object Recognition

For recognition of the image, the CNN-based deep feature extraction technique is applied. CNN takes the image and look for low-level features such as edges and curves, and then building up to more abstract concepts through a series of convolutional layers. For this research MobileNetV2 [51] is selected as a feature extractor by comparing it with others in terms of speed, accuracy and deployment infrastructure.

4.2.1.1 Feature Extraction

MobileNetV2 [51] has a set Bottleneck Residual Blocks where each has three convolutional layers. The first layer in the block is **1x1 Expansion layer** which expands the number of channels in the data before it goes into **Depth-wise convolution**. The expansion is done using a hyper-parameter called **expansion factor**, where the default is 6 [51]. The second convolutional layer is called **Depth-wise convolution** where it performs lightweight filtering by applying a single convolutional filter per input channel. The third layer is the **1x1 Projection Layer**. It projects data with a high number of channels into a tensor with a much lower number of dimensions or channels instead of only doing linear combinations of the input channel like 1x1 point-wise Convolutional Layer in MobileNetV1 [52].

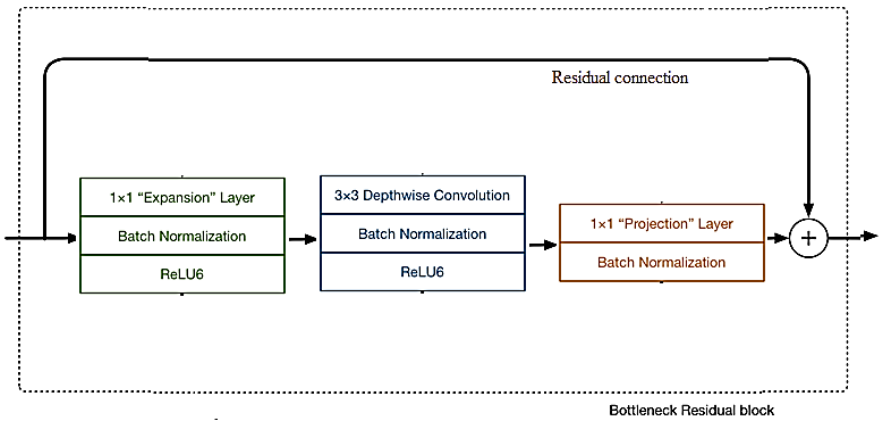


Figure 4.2 Bottleneck Residual Block

The other part of this building block is **Residual Connection** like the one in **ResNet** [53]. In residual connection, instead of each layer feeding the next layer, we skip some layers and directly feed to others layer. This is helpful to learn simple functions. As we keep increasing the number of the layer the accuracy will degrade because of vanishing gradients and curse of dimensionality.

Each layer has batch normalization and ReLU6 activation function (the projection layer doesn't have an activation function). ReLU6 is similar to ReLU, except that it prevents activations from becoming too big [51].

$$Y = \min(\max(0, x), 6)$$ Equation 1 ReLU 6 Activation Function

The following figure shows the full MobilNetV2 architecture. The network contains 1 regular 3x3 convolutional layer with 32 channels followed by 17 Bottleneck Residual Building Blocks and a regular 1x1 convolutional layer. A Global Average Pooling layer and a classification layer is added at the end of the network.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 4.3 Full MobileNetV2 Architecture

In the above figure t represents the expansion factor, c number of channels, n number of repetition and s is the number of strides.

The proposed object recognition architecture uses MobileNetV2 as a base feature extractor by removing the last layer (used as a classification layer in MobileNetV2). A Dense layer is added at the end of the feature extractor for classification. The output of the feature extractor is 1280 which will be feed into the classification layer.

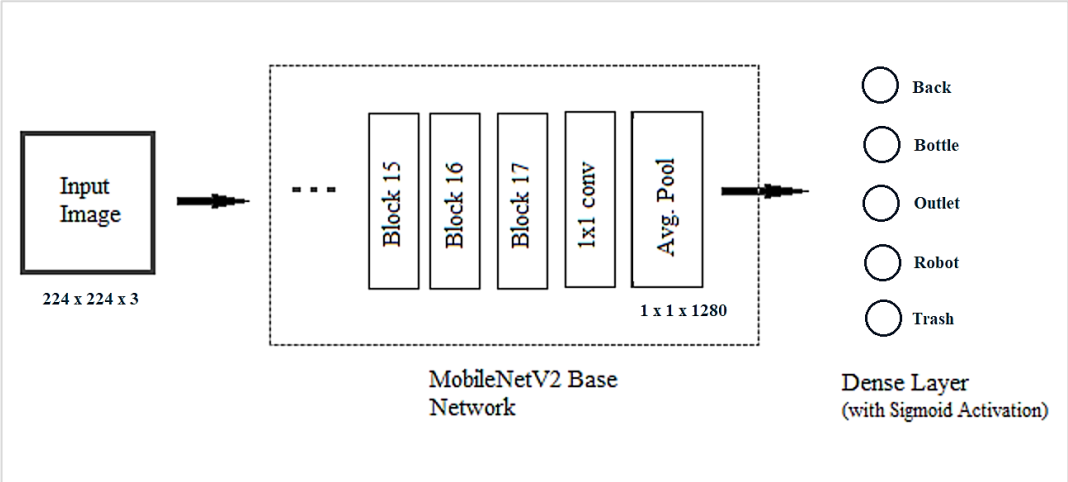


Figure 4.4 The Architecture of Proposed Object Recognition

4.2.1.2 Classification

The classification problem given in this study is a multi-class classification, where the probability that a given image is of a certain class. There are 5 classes (Background, Water Bottle, Power Outlet, EPOR Robot and Trash Can, with an index of 1, 2, 3, 4, and 5 respectively). For this particular problem given n samples

$$X = \{x_1, x_2, x_3 \ldots x_n\}$$

And labels

$$Y = \{y_1, y_2, y_3 \ldots x_n\}, \qquad \text{where } y_i \varepsilon \{1, 2, 3, 4, 5\}$$

The prediction can be estimated as

$$\hat{Y}_i = \text{argmax}_{\ j \varepsilon \{1, 2, 3, 4, 5\}} P(c_i \mid x_i), \qquad \text{where } P(c_i \mid x_i) \text{ is the probability of a class } c_i \text{ given sample } x_i.$$

For this kind of multi-class classification problem, the softmax layer is the usual choice. Softmax function takes a vector of K real numbers as input and squashes to a K dimensional

vector of real values in the range of [0, 1] that sums up to 1. The problem is that softmax does not generate the probability of a class independent of other class probability. Softmax can be used if we want to predict a single label per sample and for multi-label, we should know how many labels we need to predict for one sample. The classification problem at hand is that there could be multiple objects in one sample, so the probability for each class should be generated independently of each other.

For generating a probability for each class independently sigmoid activation function can be used. A sigmoid activation function is commonly used for binary classification, which activates each real number to another real number between 0 and 1. It uses Bernoulli distribution to generate the probability of a class c_j.

$$P(c_j|x_i) = \frac{1}{1 + \exp(-z_j)}. \qquad \textit{Equation 2 Sigmoid Function for Classification}$$

After generating the probabilities a threshold T can be used to predict whether an object of a certain class exists in the image or not.

4.2.2 Object Localization

Based on the probabilities generated on the recognition step, the objects in the images are localized using local features. For localizing the objects in the images of the scene, the feature of the object should be detected and described so that later it can be used for searching through the image of the scene.

4.2.2.1 Feature Detection

The feature of the object in an image is selected using a feature detection algorithm, which should be invariant to scale, rotation, and illumination as the robot can have a view from an arbitrary position and should be computationally efficient. For feature detection ORB (Oriented FAST and Rotated BRIEF) is selected by comparing it with other algorithms using different constraints.

ORB uses FAST feature Detector with some modifications to overcome orientation and multiscale problems. ORB feature detector has the following steps to key the key points that are scale and rotation invariant.

i. Construct a Multiscale image pyramid: ORB starts by representing the image on multiple scales, where the image is converted into a sequence of images at different resolutions. Each level in the pyramid contains an image that is down-sampled from the previous level.
ii. Compute FAST feature: ORB uses FAST feature detector to detect a potential key-points at each level in the pyramid.
    - Select a pixel P in the image. Let its intensity be I_p and a threshold T
    - Consider 16 pixels in a circle that is around P
    - Compare the intensity of p with the surrounding 16 pixels
    - Sort them into 3 groups (Lighter than P, similar to P and Darker then P) as follows

$$S_{p\to x} = \begin{cases} d, & I_{p\to x} \le I_p - t & \text{(darker)} \\ s, & I_p - t < I_{p\to x} < I_p + t & \text{(similar)} \\ b, & I_p + t \le I_{p\to x} & \text{(brighter)} \end{cases}$$

    - P is taken as a potential key-point is there are more than 8 pixels are brighter or darker than P.
iii. Apply Harris corner measure: FAST has a large response for edges since it does apply a measure of cornerness. ORB applies Harris corner measure to get top N key-points among the potential key-points detected by FAST.
iv. Assign an orientation: ORB assigns an orientation to each key-point by using the level of intensity change around that key-point. It computes the intensity centroid and constructs a vector from the corner center to the centroid.
v. Apply Non-Maxima Suppression to remove multiple detections of the same or adjacent interest points.

Pseudocode for key-point detection using ORB
Input: Image
Construct Image Pyramid

For Each Level in the Pyramid
    Select a center Pixel P
    Get Intensity of Pixel I_p and Threshold T
    For each Pixel in 16 surrounding Pixel SP
        Get the Intensity of SP, I_{SP}
        If $I_{SP} <= I_p + T$
            Store it in Darker Class
        Else If $I_p - T <= I_{SP} <= I_p + T$
            Store it in Similar Class
        Else If $I_p + T <= I_{SP}$
            Store it in Brighter Class
        End If

        If the number of pixels in brighter or Darker Class is Greater than 8
            Store It as Potential Key-point
        End If
Apply Harris Corner Measure to get Top N Key-points
For each Top N Key-points
    Assign Orientation
End If
Apply Non-Maxima Suppression
Output: Key-points

4.2.2.2 Feature Description

The proposed algorithm uses SURF (Speeded up Robust Feature) Descriptor to represent the key-points detected by FAST feature detector. SURF describes a key-point as follows

- First, calculate the Haar-wavelet responses in x and y direction within the circular neighborhood of radius 6 around the key-point using Integral Image [18].
- Calculate the Sum of vertical and horizontal wavelet responses in the scanned area.
- Change the orientation of the scanning area by $\pi/3$ and re-calculate, until the orientation with a large sum value is found.

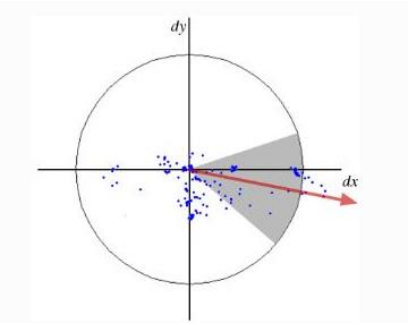


Figure 4.5 Orientation Assignment

- Construct a square 20x20 region, which is center around the key-point and oriented the above orientation.
- Split the region into 4x4 sub-regions and take a horizontal (dx) and vertical (dy) wavelet responses weighted with Gaussian ($\sigma = 3.3s$)
- The Descriptor vector

$V = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$ for each region, and with total 64 dimensions.

Pseudocode for key-point description using SURF

Input: ORB Key-points

For each Key-point in the ORB key-points

Find the orientation of Key-point

Construct a squared region around the key-point and oriented along the orientation

Split the region into 4x4 square regions

For each region

Calculate the wavelet responses (dx and dy)

Add weight with Gaussian centered at the Key-point

Add the wavelet responses and the absolute values of the responses

End For

For each region

Store the Descriptor vector

End For

End For

Output: Descriptor Vector

4.2.2.3 Bounding Box Generation

After detecting the key-points using ORB and describing it with SURF, the descriptor of the database images is matched with the image in the scene. The database images to be matched are decided by the object recognition algorithm results. The probabilities generated by the object recognition algorithm for that object is above a threshold, then the database image of that object is matched with the image in the scene.

For matching these images FLANN (Fast Library for Approximate Nearest Neighbors) based matcher is used. FLANN comes with a collection of algorithms optimized for approximating nearest neighbor fast in a large dataset and for high dimensional features. It approximates the nearest neighbors from given SURF descriptors of scene image and database image. Approximating the nearest neighbors improves the speed but it is at the cost of the algorithm not always returning the exact nearest neighbors [54].

After getting the matches from FLANN, only the best matches are selected by applying the ratio test as per Lowe [17] rule. All matches where the distance ratio is greater than T are rejected, where T is a threshold chosen empirically. The number of accepted matches should be greater than the threshold to decide it is a match. The accepted matches are once again sorted and top N matches are selected for calculating the bounding box. For calculating the bounding box, the key-point of the scene image is extracted from the Top N matches. The bounding box is calculated as follows

BB = (Min (x_n), Min (y_n), Max (x_n), Max (y_n)), where x_n is x-axis and y_n is y-axis of the key-point.

Pseudocode for bounding box generation

Input: Class probability

For Each Probability p

If P is greater than T

Detect Feature of Scene Image

Describe the Scene Image (DS)

Detect Feature of Database Image

Describe the Database Image (DD)

Match DS and DD using FLANN

Apply a ratio test

If Matches > Threshold

Sort the Matches

Select Top N

Calculate the bounding box

Store the bounding box

Else

Not enough number of Matches

End If

End If

End For

Output: bounding boxes

4.3 Mobile Robot Navigation

Mobile robot navigation is important for a mobile robot to move from one point to another, avoiding situations like collision and unsafe conditions. The proposed Mobile Robot Navigation uses ROS Navigation Stack with Turtlebot3, which supports SLAM. When building or navigating a map, SLAM ignores relevant descriptive information of the environment like what kind of object it contains. The proposed navigation is based on SLAM and object detection algorithm. The map creation, navigation process and how the object detection algorithm is integrated with navigation stack for navigation are described in the following subsections.

4.3.1 SLAM Process

In order to create a map the distance value of the object from certain objects are required and these values can be scanned using sensors and Depth camera from the XY plane. The second important value is the pose (position and orientation) value which depends on the odometry of

the robot. The distance measured (also called Scan in ROS) using LIDAR and the pose information which is affected by the relative coordinate (also called TF (transform) in ROS) are the two pieces of information that are used to create a Map.

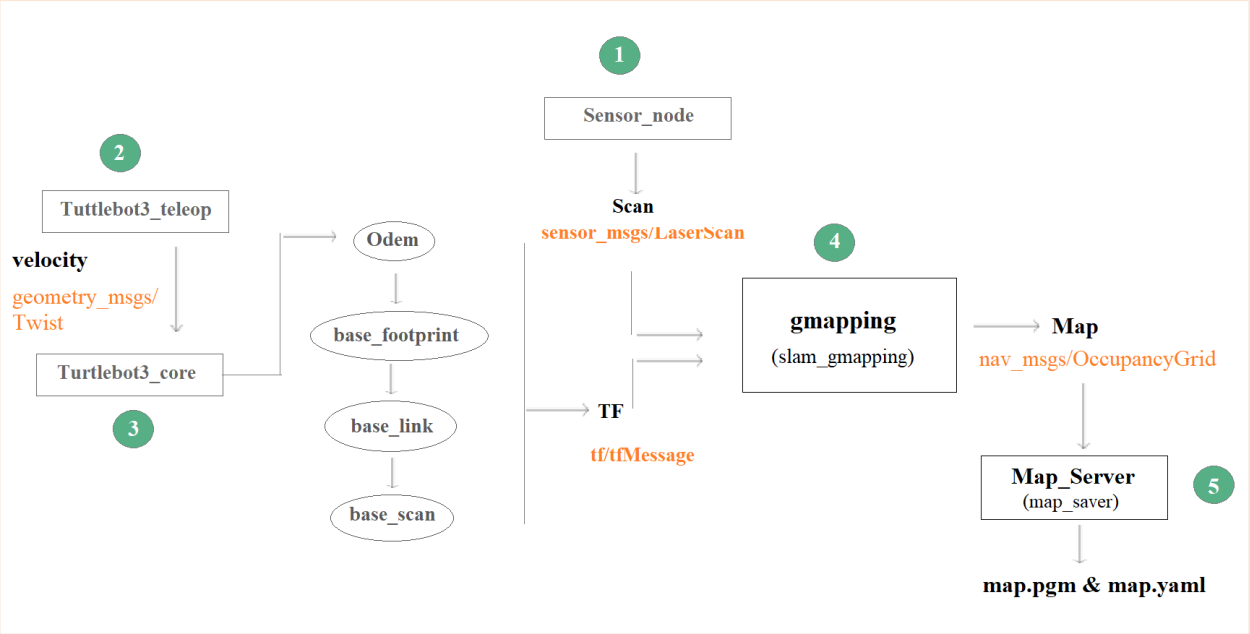


Figure 4.6 Flow of SLAM Procedure

As shown in the above diagram sensor node runs the LIDAR sensor and sends the scan information to the gmapping node, which is used for SLAM. The "turtlebot3_telop" node controls the robot by using the keyboard input and sends the translation and rotation speed command to the "turtlebot3_core" node. The robot is moved using the translation and rotation speed command relative coordinate of odometry in "tf" from is published by "turtlebot3_core" node.

The map is created by "turtlebot3_slam_gmapping" node based on the scan information from the LIDAR sensor and the tf (pose value of the sensor). The created map is saved by "map saver" node in the map server package.

4.3.2 ROS Navigation Stack

ROS provides a Navigation stack which is designed to be general-purpose as possible. It uses information from odometry and sensor streams and outputs velocity commands to a mobile base. The Navigation stack is responsible for creating a map of the environment using SLAM,

measure and estimate robots pose, looking out for obstacles using distance sensors like laser-based, Ultrasonic, infrared or Vision-based distance sensors, and for path calculation and driving.

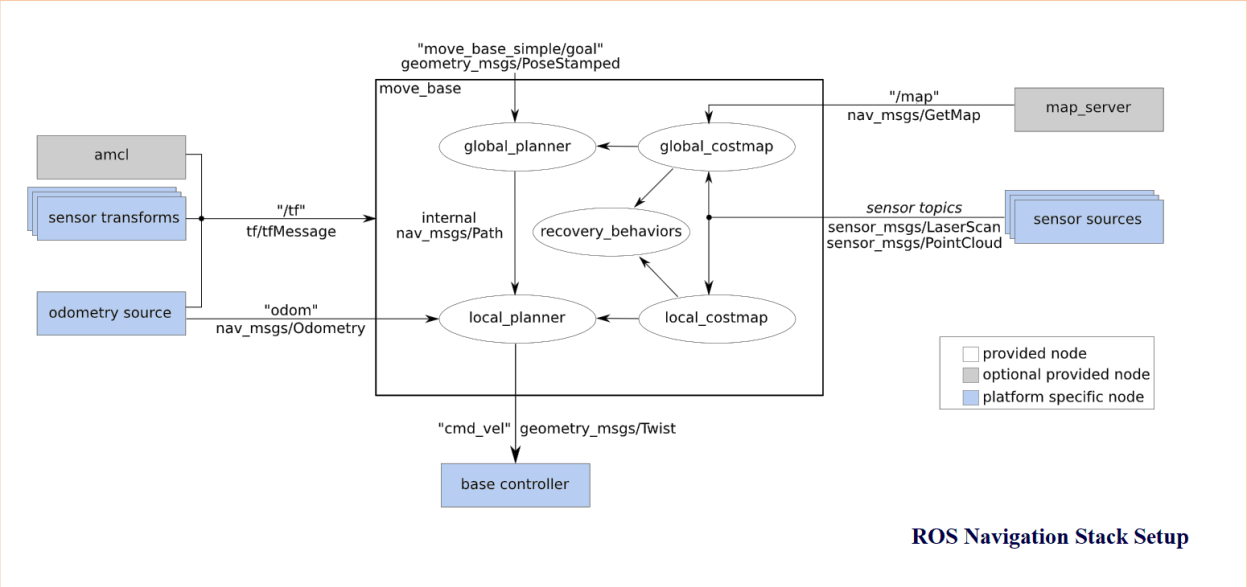


Figure 4.7 The Relationship Between Essential Nodes and Topics for ROS Navigation Stack [47].

The above figure illustrates different essential nodes that are required for navigation. The distance value measured using a sensor like LIDAR is used to estimate the robot's current pose or the motion of the robot using Adaptive Monte Carlo Localization (AMCL). The pose of the robot can change depending on how the hardware of the robot is configured. ROS uses relative coordinate transformation (TF) to obtain the coordinate of the sensor.

The map created using SLAM is loaded from the map server to calculate the cost-map. The map, pose of the robot and sensor and obstacle information are used to load the static map and utilize the occupied area, free area and unknown area for navigation. Cost-map used in navigation calculates the obstacle area, possible collision area, and movable area based on the map, pose of the robot and sensor and obstacle information. Two kinds of cost-maps are created, global cost-map and local cost-map. The global cost-map is used for navigating the global area where the local one is used path planning and obstacle avoidance in a limited area of the robot.

The path planning is done by Move Base node using the pose of the robots, cost-map, and topics from sensors. The Odometry information of the robot is used for local path planning and

avoiding obstacles. After planning a path, Move Base publishes velocity commands for a Base Controller to navigate to the destination.

4.3.3 Object Detection Integration with a Navigation System

When using SLAM to navigate through the environment while simultaneously building a map, the descriptive information like knowing what object exists in the environment is ignored. We have the position and orientation of the robot and the sensors, obstacle information and the map which describes the moveable area, obstacle area and possible collision areas from the above process. Even though this information can tell us where the obstacles are and where are the moveable and possible collision areas, it doesn't provide information like what are the objects that the environment contains and what is that place. Representing the environment in terms of objects it contains, gives a better and accurate model of the environment and it allows an exchange of information between robots or between robots and humans semantically.

Semantic representation of the environment for navigation is called Semantic Navigation. The object detection algorithm is integrated with navigation to represent the environment in terms of the objects it contains. The mobile robot navigation has two steps, which are building a map to represent the environment and navigating through the map, while simultaneously mapping the environment. Three nodes are added. The first one is for building a map together with locating objects in the environment using object detection. The second one is for navigating from one point to another while simultaneously updating the map and object location. The last one is for interacting with the robot by delivering commands.

As shown in the following mapping stage figure, "Object_mapping" node is responsible for detecting objects in the environment and adding it to the map. The gmapping node takes the laser scan and pose of the robot and creates a map. All objects around the robots are considered as an obstacle in gmapping. "Object_mapping" node subscribes "PI_camera" node to receive a frame from PI camera and classifies that frame using object recognition, and localizes objects in that frame based on the probability generated by object recognition. After detecting the object in the frame it registers that object to that location. It also publishes the image with the bounding boxes around the object for displaying it to the screen. The map created by SLAM gmapping is saved by the map server for later navigation.

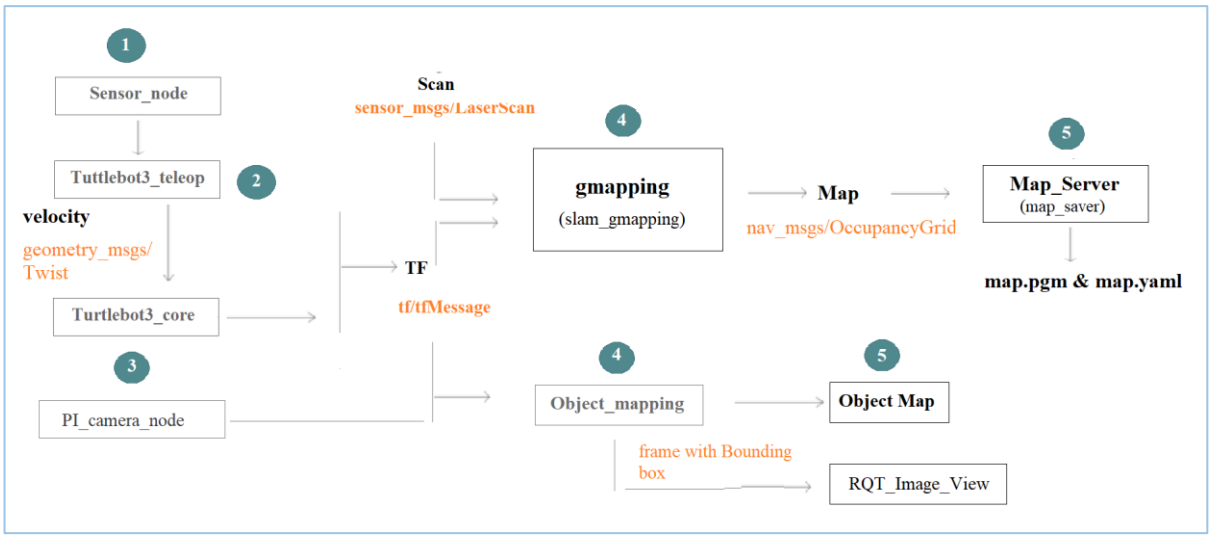


Figure 4.8 Map Creation Process for Navigation

The second stage is navigating using the map created by the above process. For navigation "Nav_Object" node and "Command_node" are added. "Command_node" accepts the navigation command (destination in semantic) from the user (human) and create's and executes navigation goals to the destination. "Nav_Object" node is responsible for detecting objects along the way and updating the object map.

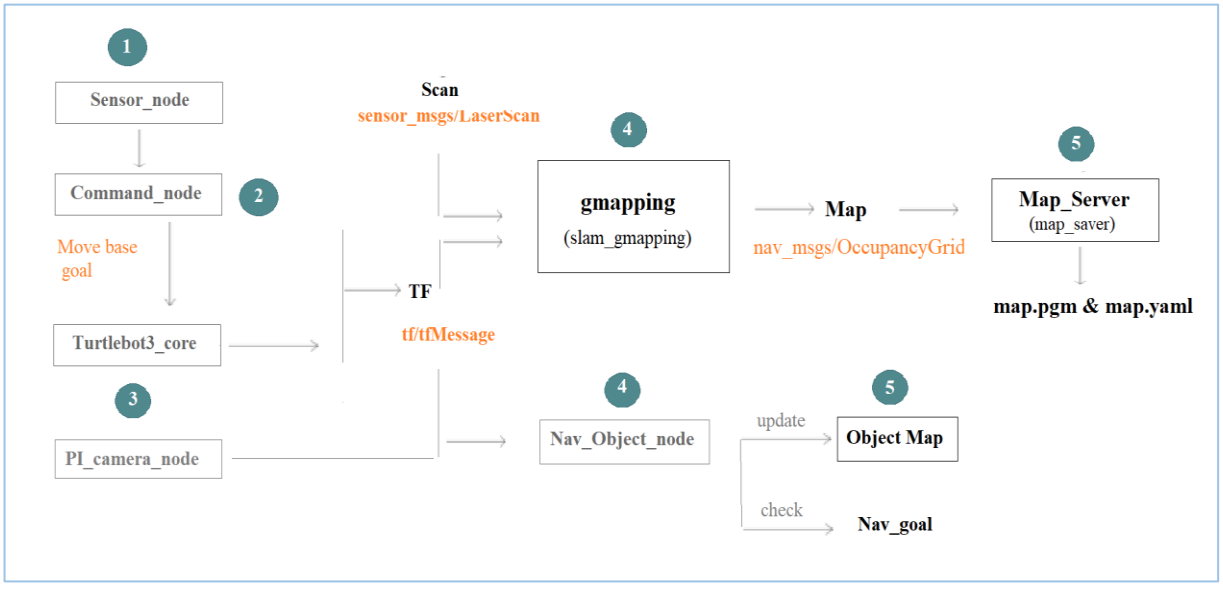


Figure 4.9 Navigation Process Using the Object Detection

Once the robot reaches the destination this node is responsible to confirm the location by looking for the destination object.

When the object detection algorithm is integrated with the Turtlebot3 for navigation, the distance information is extracted from the LIDAR mounted on the top of the Turtlebot3. 360 Laser Distance Sensor LDS-01 is used. The LIDAR is a 2D laser scanner capable of sensing 360 degrees around the robot. The object detection algorithm node subscribes to scan topic to get the 360 distance data from the robot to the object.

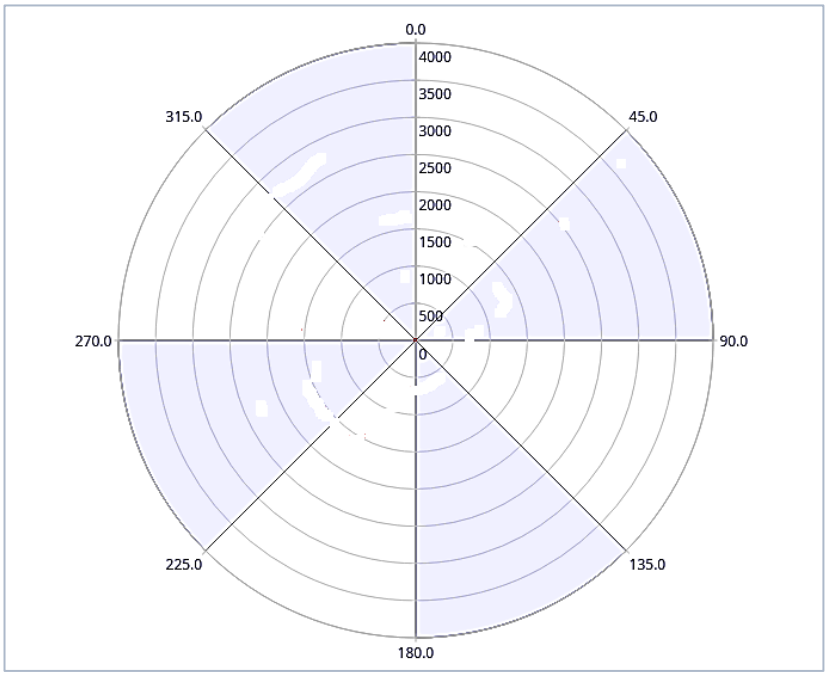


Figure 4.10 LDS range and distance

The scan starts from the front (0 degree) and rotates 360 degrees. The camera is mounted on the front of the robot, so the distance of the object from the robot is in a range of 350 to 10 degrees.

CHAPTER FIVE

5 Implementation of the Proposed Solution

5.1 Overview

In this chapter, the implementation of the proposed solution is described. The implementation of an object detection algorithm, mobile robot navigation system and the integration of object detection algorithm for mobile robot navigation, prototype setups are included.

5.2 Prototype Development Setup

5.2.1 Working Environment

- Desktop: The desktop computer is used for developing mobile robot navigation
    - Operating system: Ubuntu 16.04 LTS
    - Processor: Intel ® Core™ i5-6500 CPU @ 3.20GHz x 4
    - Graphics: Intel ® HD Graphics 530 (Skylake GT2)
    - Installed memory (RAM): 4.00 GB (3.7 GB usable)
    - System Type: 64-bit Operating System, x64-based Processor
- ROS

ROS kinetic kame 2016 is installed on the Desktop computer and Turtlebot3. The Desktop version acts as a master, where the turtlebot3 version acts as a servant.

- Laptop: the laptop computer is used for object detection algorithm development.
    - Operating system: Window 10
    - Processor: Intel(R) Core™ i5-6200U CPU @ 2.30GHz 2.40 GHz
    - Installed memory (RAM): 4.00 GB (3.71 GB usable)
    - System Type: 64-bit Operating System, x64-based Processor
- Visual Studio Code

Visual Studio Code is used as a development IDE, with python interpreter 2.7 on the desktop computer and 3.6 on the laptop desktop.

- Google Colab is used for training and testing the deep neural network for multiple object recognition.

5.2.2 Turtlebot3 Burger Robot Specification and Setup

Turtlebot3 burger comes with 360 degrees Laser Distance Sensor LDS-01 which is capable of sensing 360 degrees with detection Distance of 120mm ~ 3500mm for SLAM and Navigation. This LiDAR is mounted on top of the robot and works by calculating the difference of the wavelength when the laser source is reflected by the object. The object around the robot should be reflective since it measures the return of a light source. Ubuntu Mate 16.04 operating system and ROS are installed on Raspberry Pi 3 with 16GB Storage.

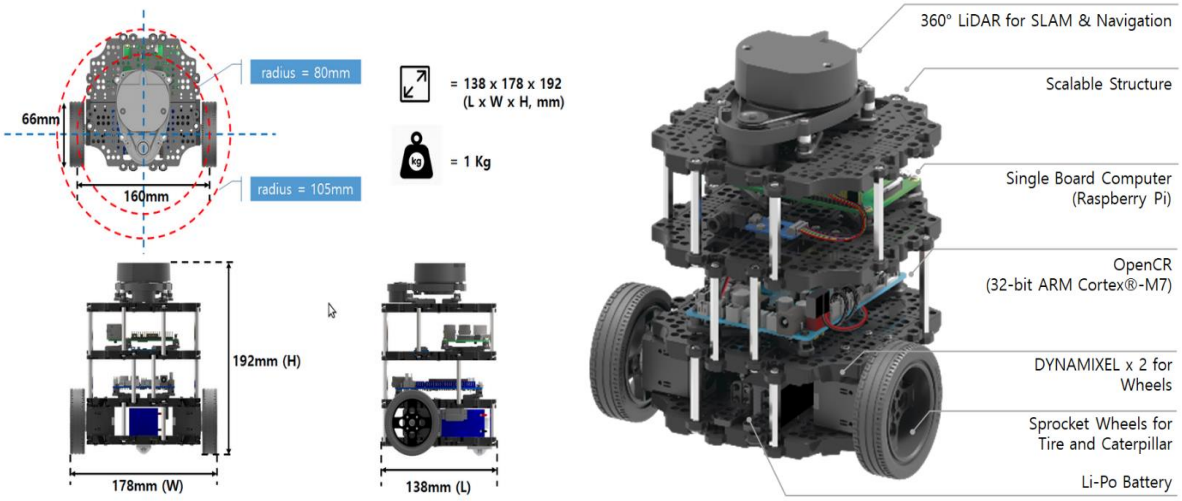


Figure 5.1 Turtlebot3 Burger Specification

Raspberry PI Camera Module v2 with Sony IMX219 8-megapixel sensor is integrated with a robot for the high definition video feed. It has 2 DYNAMIXEL wheels from the back for moving and calculating the odometry. The Raspberry PI 3 and Arduino board is used to control the robot and Raspberry PI Camera is used to feed images for object detection algorithm.

5.2.3 Implementation Environment

An indoor test-bed is designed for the purpose of mobile robot navigation in an indoor environment. The test best represents an indoor computer laboratory environment with some tables (white boxes), trash can, a power outlet, Water bottle, and EPOR robot on the floor. The image dataset is also collected from this test-bed.

Figure 5.2 Test-bed for Mobile Robot Indoor Navigation

5.2.4 Dataset Description

The dataset contains images of 5 classes including the background class. These classes are Trash Can, Power Outlet, EPOR Robot, Water Bottle, and Background. The objects are limited to 4 because of the Turtlebot3 Robot height. The robot height is 192mm, so the object should be in the sight of view. The dataset contains multiple images of an object from different angles, scale, illumination, and view.

Water Bottle	
Power Outlet	

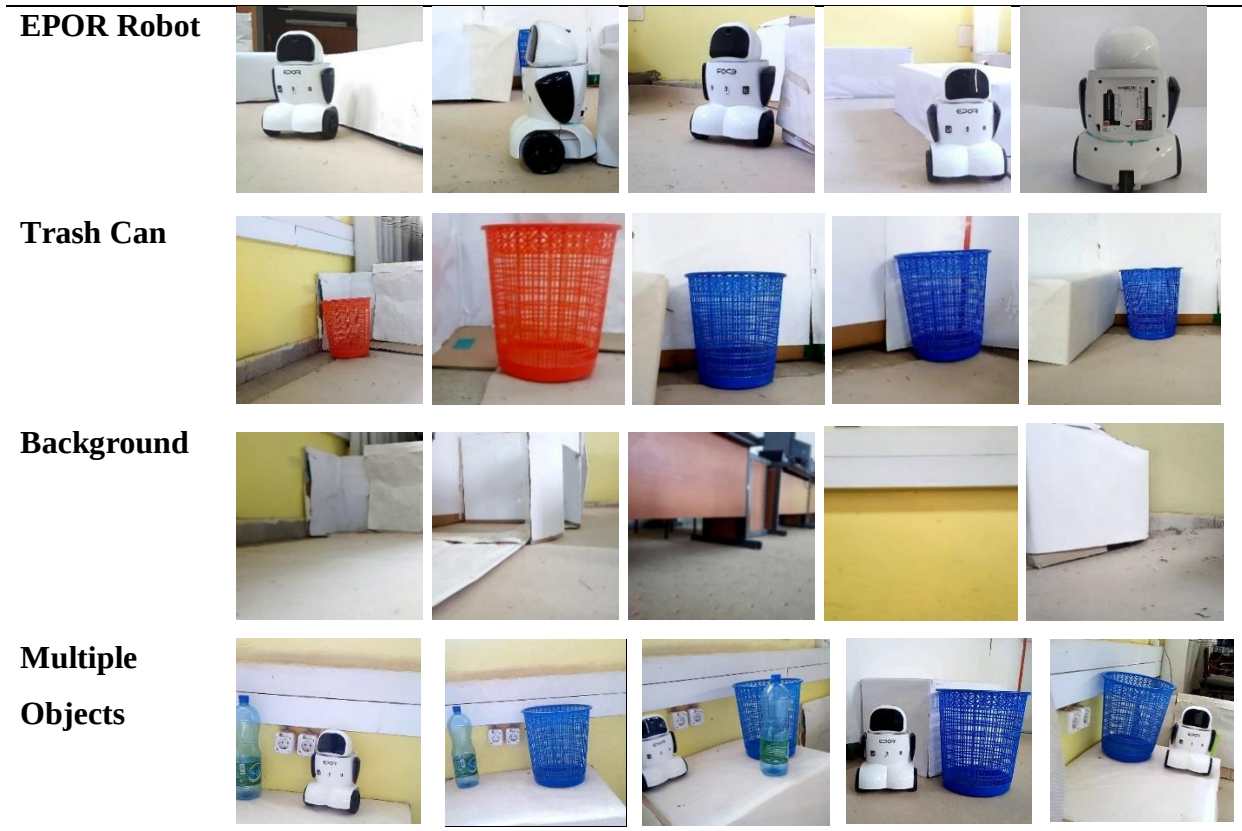


Figure 5.3 Sample Image Dataset

The images are taken at different background, scale, view, angle, and illumination to cover all possible scenario that a robot may operate.

Table 5.1 The Number of Examples for Each Class.

Class	**Number Of Images**
Water Bottle	1830
Power Outlet	1830
EPOR Robot	2048
Trash Can	1830
Background	1968
Multiple objects	5717
Total	15223

5.3 Object Detection

For implementing the proposed object detection algorithm the following library and frameworks are used. For preparing the training and testing dataset and for implementing the local feature-based object localization OpenCV 3.3 is used. For creating and training the deep learning model Tensorflow 1.13 and Keras 2.2.4 are used.

5.3.1 Object Recognition Implementation

The first stage in the proposed object detection algorithm is classifying the image in the scene by generating a probability that the image belongs to that class. Since object recognition is based on deep CNN features it is implemented using two deep learning frameworks, Tensorflow and Keras. To create a model for multiple object recognition Keras with Tensorflow backend is used. The model is trained using Google Colab and tested using both the laptop and Google Colab. Since training a deep learning model takes a lot of time a GPU is required. Google Colab offers a 12 GB RAM with GPU and more than 135 GB storage space for free.

All the image dataset is uploaded to Google Drive and then mounted with Google Colab to access the images while training and testing. MobileNetV2 Keras implementation has above 3.5 million parameters. The last logits or classification layer has about 1.3 million parameters. In the proposed algorithm the MobileNetV2 is used as a feature extractor by removing this classification layer and adding another classification layer with fewer neurons. The proposed architecture has 2.3 million parameters that are pre-trained on ImageNet. The dataset images are resized to 224 x 224 x 3 before they are fed into the model.

5.3.2 Object Localization Implementation

Object localization step is responsible to find and localize the object recognized in the image of the scene. OpenCV is used to implement the feature detection and description with ORB and SURF. The feature of the image of the scene and database image of each object is extracted and described using ORB and SURF respectively. The database images of each object are loaded and the feature of each image is extracted and described.

```
1    img2 = cv2.imread (dir_path + 'bottle.jpg')    # Database Image For: Bottle
2    img3 = cv2.imread (dir_path + 'outlet.jpg')    # Database Image For: Outlet
3    img4 = cv2.imread (dir_path + 'robot.jpg')     # Database Image For: Robot
```

```
4       img5 = cv2.imread (dir_path + 'trash.jpg')      # Database Image For: Trash
5
6       self.dataImages = [img2, img3, img4, img5]
7
8       kp2 = self.orb.detect (self.dataImages [0], None)
9       kp3 = self.orb.detect (self.dataImages [1], None)
10      kp4 = self.orb.detect (self.dataImages [2], None)
11       kp5 = self.orb.detect (self.dataImages [3], None)
```

The feature of each object is detected by ORB feature detector (8 – 11). The detected key-points are described using SURF descriptor (12 – 15). The detected key-points with its description are added to an array so that it can be accessed to match objects in the scene.

```
12       kp2, des2 = self.surf.compute (self.dataImages[0], kp2)
13       kp3, des3 = self.surf.compute (self.dataImages[1], kp3)
14       kp4, des4 = self.surf.compute (self.dataImages[2], kp4)
15       kp5, des5 = self.surf.compute (self.dataImages[3], kp4)
16
17       self.keyp = [kp2, kp3, kp4, kp5]
18       self.desc = [des2, des3, des4, des5]
```

The probabilities generated by the model in the recognition step is sorted based on its probability and the one with probability is higher than the threshold is selected.

```
18      maxz = heapq.nlargest (5, range (len(Prediction)), out.take)
19
20        for i in range (5):                              # since the number of classes are 5
21           max1 = maxz[i]
22
23             if (out[max1]>0.6 and max1 != 0):          # if the probability is greater than 60%
```

The feature of the Image of the scene is detected and described using ORB and SURF respectively and matched with the selected database image using FLANN with k value 2.

```
24                  kp1 = self.orb.detect (img1, None)
25                  kp1, des1 = self.surf.compute (img1, kp1)
26
27                  theMatch = self.flann.knnMatch (dec2, des1, k=2)
```

The match is sorted with respect to its distance so that later the top 10 can be selected. FLANN finds the approximate nearest neighbor by calculating the distance between the descriptors. In order to get good matches the Lowe ratio test is applied. The threshold of 0.85 is selected empirically by trying different values.

```
28              theMatch = sorted (theMatch, key = lambda x:x[0].distance)
29            goodF = [m1 for (m1, m2) in theMatch if m1.distance < 0.85 * m2.distance]
```

When the number of good matches is greater than the threshold the top 10 are selected. The location of each matched key-points is extracted from both images and the homography matrix is calculated along with the mask. The outliers are removed by the RANSAC algorithm so the mask on contains the inliers.

```
30          if len(goodF)>MIN_MATCH_COUNT:
31
32              goodF = goodF[:10]
33
34               src_pts = np.float32([ kp2[m.queryIdx].pt for m in goodF ]).reshape(-1,1,2)
35              dst_pts = np.float32 ([ kp1[m.trainIdx].pt for m in goodF ]).reshape(-1,1,2)
36                 M, mask = cv2.findHomography (src_pts, dst_pts, cv2.RANSAC, 2)
37              matchesMask = mask.ravel().tolist()
```

Once only the mask is found, all inlier can be extracted from the good matches and the bounding box can calculated using those key-points.

```
39              o = 0
40              allx = []
41               ally = []
42              for m in goodF:
43                 if(matchesMask[o] == 1):
44                    allx.append(kp1[m.trainIdx].pt[0])
45                    ally.append(kp1[m.trainIdx].pt[1])
46                 o = o + 1
47
48                 minx = min(allx)-10
49                 miny = min(ally)-10
50                 maxx = max(allx)+10
51                  maxy = max(ally)+10
52
53                  xcenter = (maxx + minx) / 2
54                  ycenter = (maxy + miny) / 2
```

minx and miny, and maxx and maxy represents the top-left corner and bottom-right corner of the bounding box respectively.

5.4 Mobile Robot Navigation

Mobile robot navigation includes building a map while detection objects and mapping them, and navigation using the created map while simultaneously mapping, and detecting objects and updating the map.

5.4.1 Map Building Implementation

The map of the indoor test-bed is used as an initial map in the navigation stage. One node called "Blackmap" is added for map building. This node subscribes to three topics (Odem, scan and camera). This node receives frames from a camera node to detect objects using the object detection algorithm and map it. The distance to an object is extracted from the LiDAR scan by subscribing to the scan topic. The Odom topic is used to get the position and orientation of the robot, which will be used to know the location of the robot when it detects an object. After detecting objects in the frame the node publishes the frame so that it can be displayed using RQT.

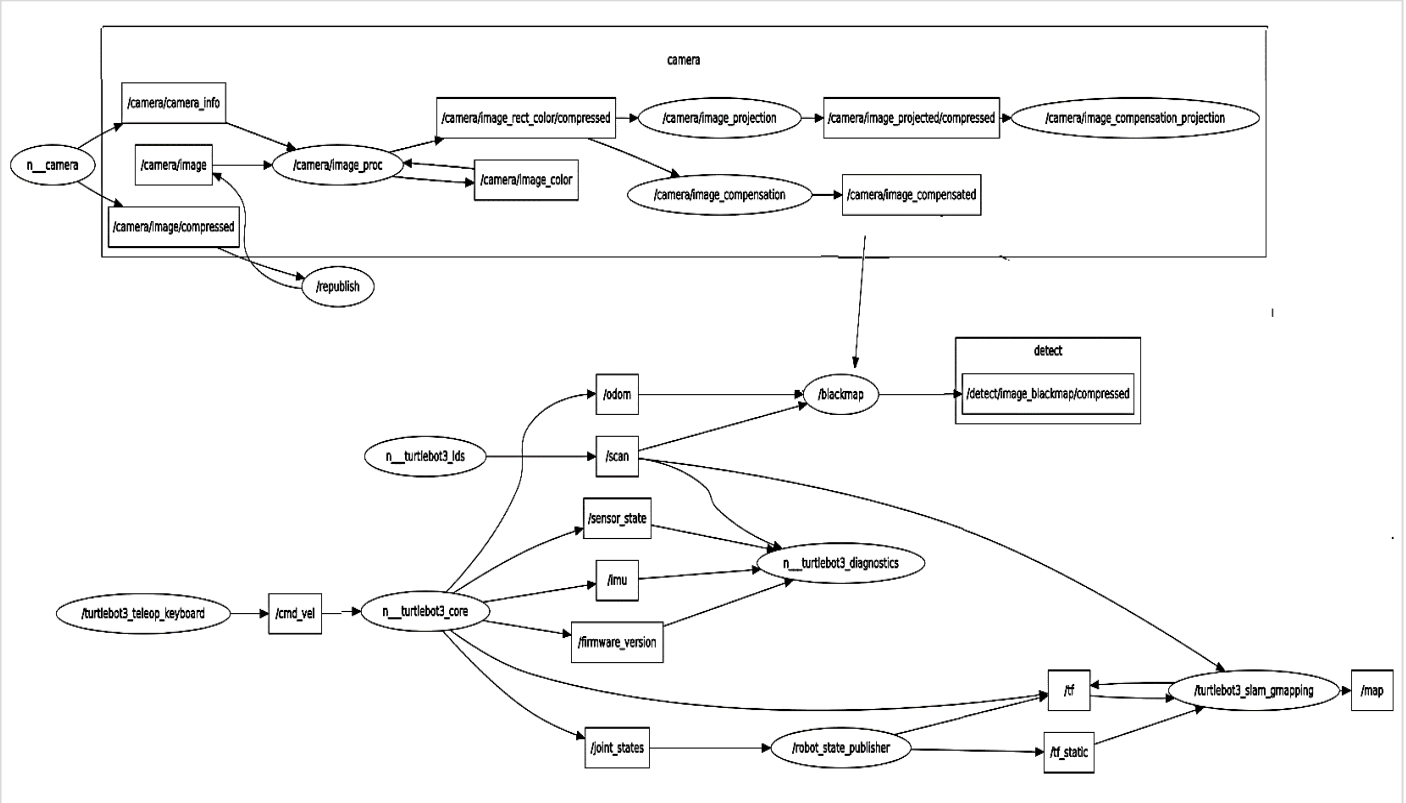


Figure 5.4 Mapping Graph Created by RQT

As shown in the above diagram the Turtlebot movement is controlled by turtlebot3_teleop_keyboard node. The Scan and TF information is subscribed by the turtlebot3_slam_gmapping node to map the environment.

5.4.2 Navigation Implementation

Navigation can be implemented using the environment map and object map created by the mapping process as an initial step. While navigating the environment it maps the environment using SLAM and also detects objects using the object detection algorithm. This time the robot is controlled by user (human) command using text. So for navigation, another two nodes are added, "blackcommand" and "blacknav". Users command the robot using "blackcommand" node and this node creates a navigation goal and executes as a user command. "blacknav" node does a similar job as "blackmap" node, it subscribes Odem and Scan topic for knowing the pose of the robot and distance to an obstacle.

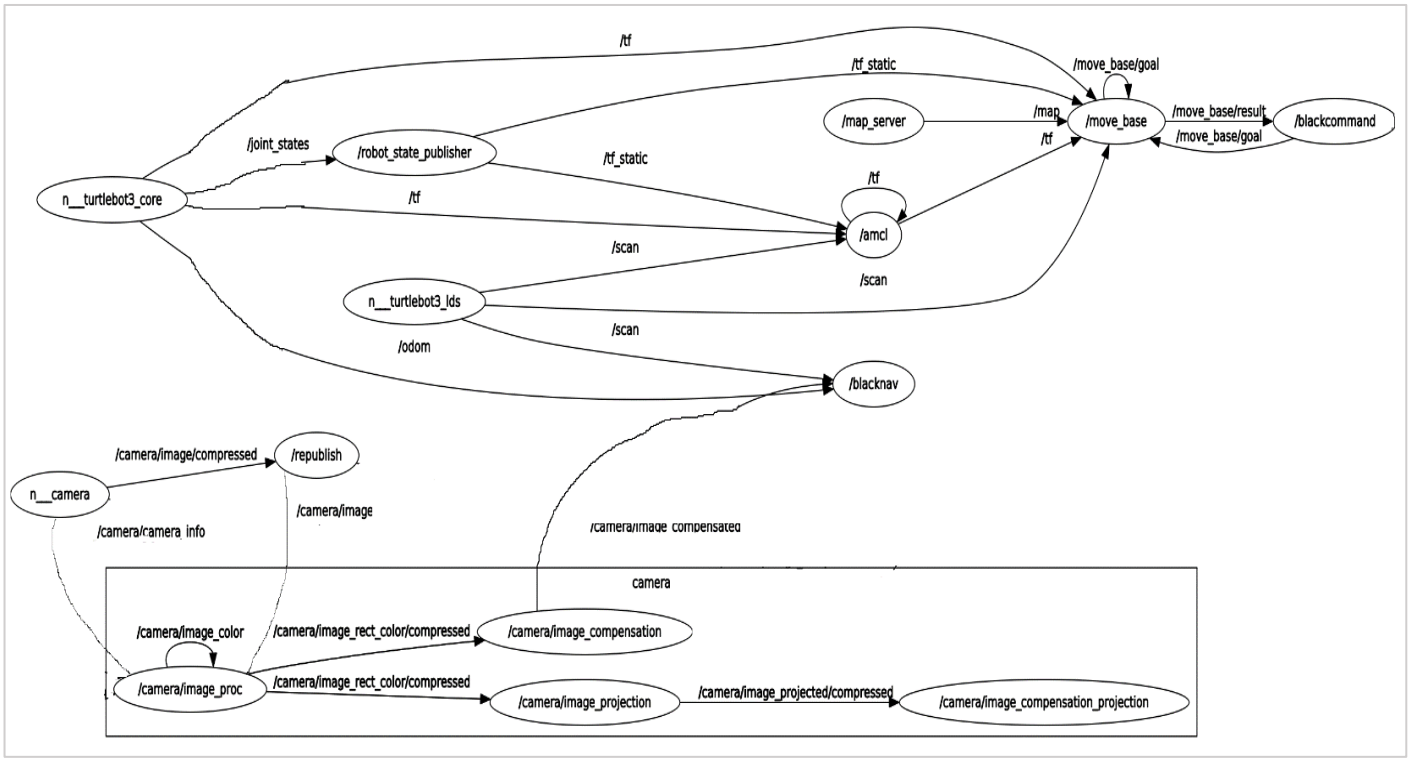


Figure 5.5 Navigation Graph Created by RQT

While "blackcommand" node is executing the navigation goal, the "blacknav" node detects an object and maps it and the turtlebot3_slam_gmapping maps the environment.

CHAPTER SIX

6 Evaluation, Results, and Discussion

6.1 Overview

This chapter presents the evaluation of the object detection algorithm and the integration of object detection with mobile robot navigation. It also discusses the result by comparing it with other related works.

6.2 Object Detection

6.2.1 Object Recognition

The object recognition algorithm takes an image from the scene as an input and generates the probability that the input image belongs to each class. Each class probability is calculated independently using a sigmoid function. The image dataset contains 15,018 images from the computer vision lab and test-bed. This set is split into training and test dataset for training and testing the deep learning architecture. The dataset contains images of one object or multiple objects. The training process is split into two steps. First, the model is trained on the images that contain only one object using different parameters, so that it can specialize in recognizing each object. Then the best performing model is trained on the whole dataset (on the image containing both one object and multiple objects).

For cross-validating the model, the Holdout cross-validation technique is used. In Holdout cross-validation, the dataset is randomly split into training and test set. The training set is used for training the model and the test set is used for validating the trained model. This technique works well when the data is not limited and the training and test sample has the same distribution. The parameters used for both training are the amount of training and test split, and the learning rate. The batch size used is 32, and the number of epoch is 500 for the first and 1000 for the second step. The number of epoch is chosen empirically and early stopping is used to avoid overfitting.

In the first step different combination of train-test split and learning rate is used to train the model using 9,367 images (each contains only one object). The train-test splits used are 70/30, 75/25, 80/20, 85/15, 90/10 and 95/5, and learning rates are 0.0001 and 0.00001. Other train-test split and learning rate values are also used to check if there are more efficient parameters are

there. The total number of training in the first step by combining the above parameter is 10. The following tables summarize the training and testing result for different train-test split and learning rates.

Table 6.1 Training and Testing Report for the Learning Rate of 0.00001

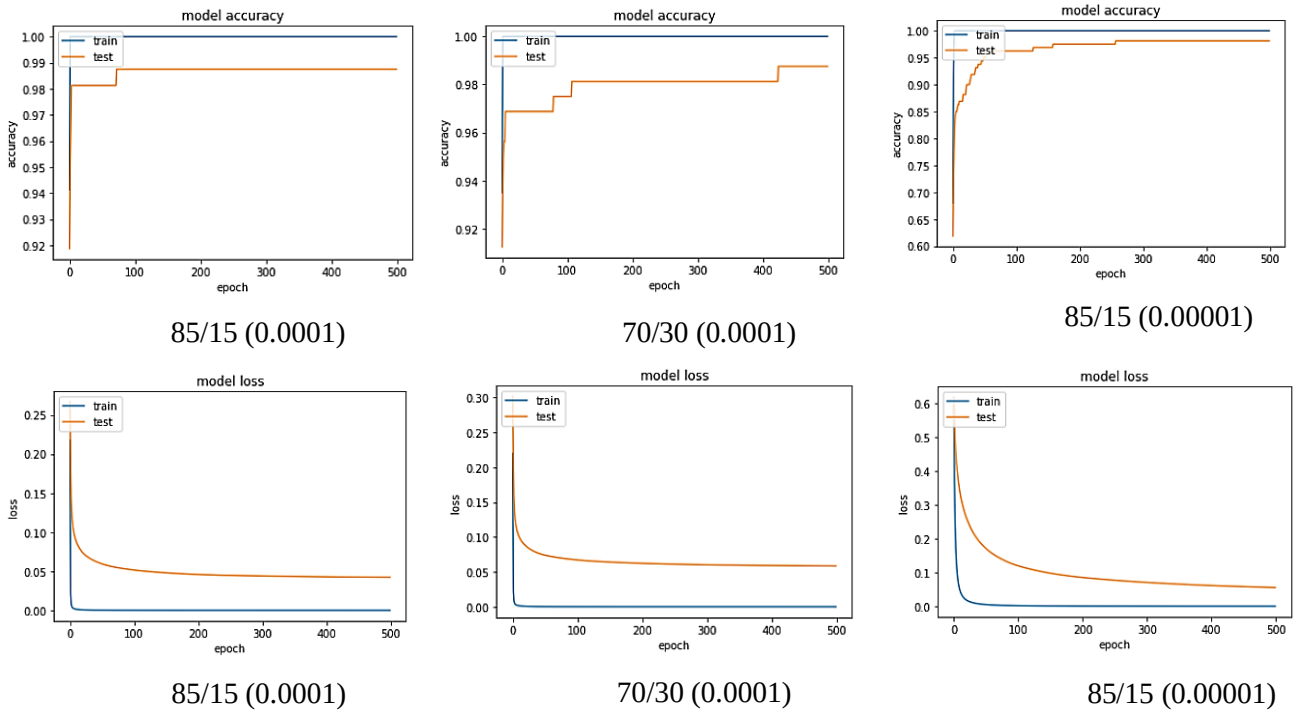
LEARNING RATE	TEST SIZE (%)	NO OF TEST	NO OF TRAIN	TRAIN LOSS	TRAIN ACC	TEST LOSS	VALID ACC	TEST ACC	PRE	RECALL	F1
-4	30	2811	6556	2.60E-06	1	0.06	0.9875	0.98	0.98	0.98	0.98
-4	25	2342	7025	2.40E-06	1	0.02	0.9875	0.973	0.974	0.973	0.97
-4	20	1874	7493	2.20E-06	1	0.075	0.968	0.953	0.954	0.953	0.95
-4	15	1406	7961	2.30E-06	1	0.04	0.9875	1	1	1	1
-4	10	937	8430	1.90E-06	1	0.038	0.98	0.94	0.94	0.94	0.93
-4	5	469	8898	2.30E-06	1	0.05	0.97	0.97	0.96	0.96	0.96

Table 6.2 Training and Testing Report for the Learning Rate of 0.0001

LEARNING RATE	TEST SIZE (%)	NO OF TEST	NO OF TRAIN	TRAIN LOSS	TRAIN ACC	TEST LOSS	VALID ACC	TEST ACC	PRE	RECALL	F1
-5	30	2811	6556	7.20E-05	1	0.08	0.975	0.97	0.97	0.97	0.97
-5	25	2342	7025	6.90E-05	1	0.05	0.9875	0.953	0.954	0.953	0.95
-5	20	1874	7493	5.90E-05	1	0.05	0.9875	0.953	0.955	0.953	0.95
-5	15	1406	7961	6.50E-05	1	0.055	0.98	0.98	0.98	0.98	0.91
-5	10	937	8430	5.50E-05	1	0.774	0.975	0.93	0.934	0.93	0.93
-5	5	469	8898	6.70E-05	1	0.08	0.96	0.93	0.94	0.93	0.93

The number of training images ranges from 6556 to 8898 depending on the train-test split. The training accuracy for all the training is 100% even though the training loss varies. The highest validation accuracy achieved is 0.9875 (which is 99%). The training with a learning rate of 0.0001 has higher accuracy than the training with a learning rate of 0.00001 in most cases. The lower the learning rate, the better it learns and the more data it requires. In this case, with given data, the model learns better with a higher learning rate. To evaluate the learning process the training history of the top 3 accurate models is plotted. The top three are the ones with a

parameter of 85/15 train-test split and 0.0001 learning rate, 70/30 train-test split and 0.0001 learning rate, and 85/15 train-test split and 0.00001 learning rate



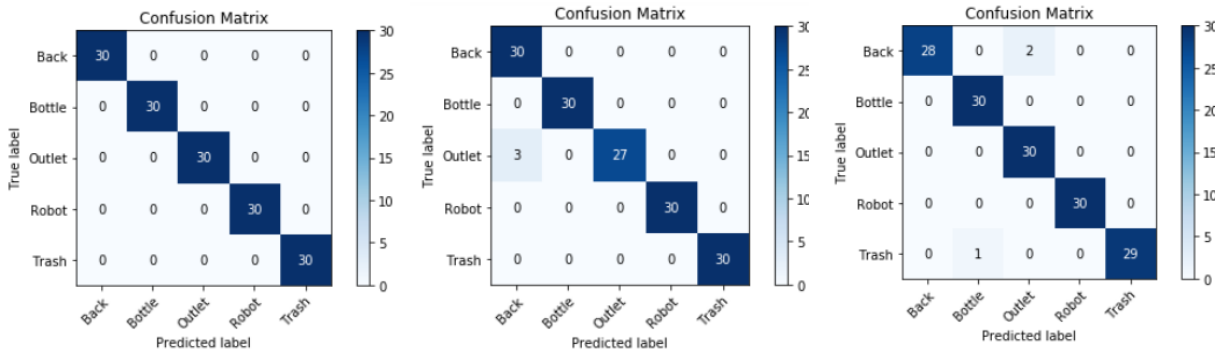
85/15 (0.0001) 70/30 (0.0001) 85/15 (0.00001)

85/15 (0.0001) 70/30 (0.0001) 85/15 (0.00001)

Figure 6.1 Training Summary for Top 3 One Object Models

As shown in the above figure the model with a learning rate of 0.00001 has a smooth learning curve. It starts with low accuracy or high loss and the accuracy increases or the loss decreases slowly. The first model has the highest test and training accuracy and the lowest test and train loss. The middle one is the second most accurate and the last one is the third most accurate.

The performance of the trained model is evaluated using the test set (150 images containing one object). The confusion matrix of the top 3 accurate models is presented in the bellow figure. The first one (85/15 and 0.0001) is 100% accurate on a given test set since it predicted all images correctly. Unlike the first one, the middle one (70/30 and 0.0001) predicted 3 images that are outlets as background. The last confusion matrix represents the 85/15 train-test split and 0.00001 learning rate, and it predicted 2 images that are background as an outlet and 1 image that is trash as a bottle.



85/15 (0.0001) 70/30 (0.0001) 85/15 (0.00001)

Figure 6.2 Testing Confusion Matrix for the Top 3 One Object Models

The model that is trained with an 85/15 train-test split and 0.0001 learning rate are chosen as a feature extractor for generating the probability that an object exists in the image. If there are multiple objects in the same image, the model should recognize each object. Since the chosen model is able to discriminate each object, it is trained on images that contain both only one object and multiple objects. The training and testing summary are given in the following table.

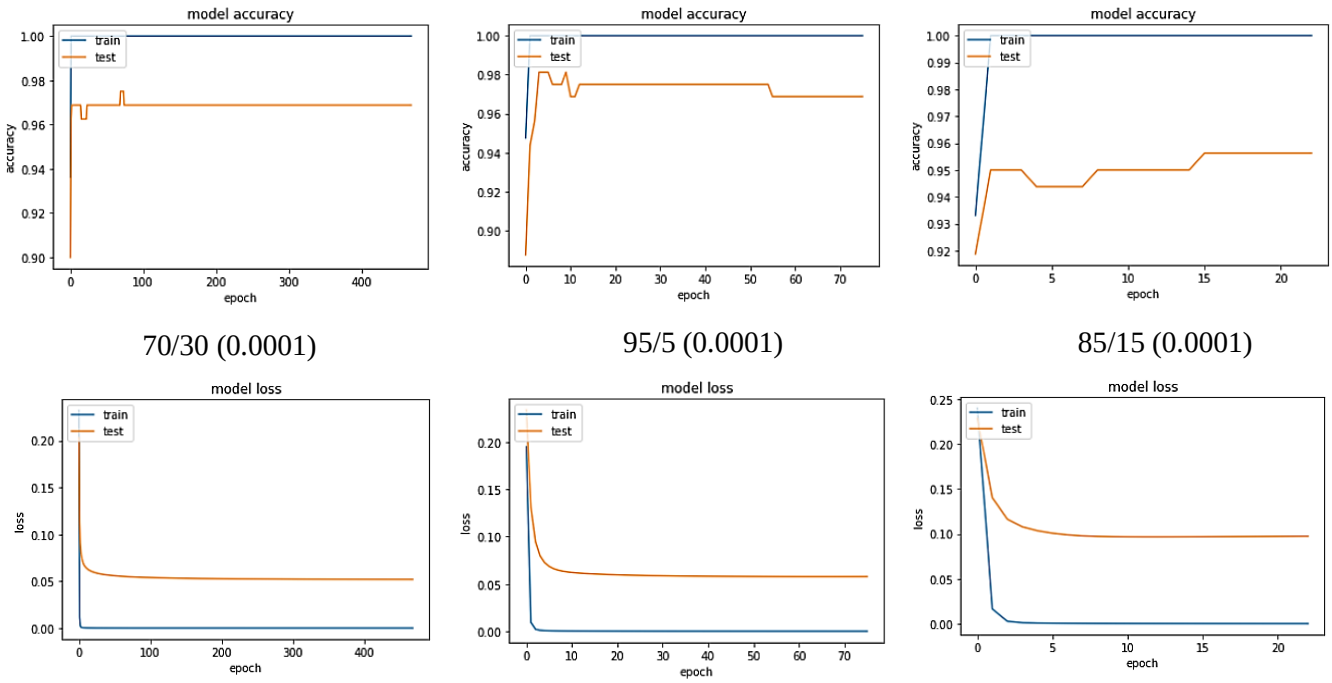
Table 6.3 Training and Testing Report for Learning Rate of 0.0001 (main model)

LEARNING RATE	TRAIN TEST SPLIT	NO OF TEST	NO OF TRAIN	TRAIN LOSS	TRAIN ACC	TEST LOSS	VALID ACC	TEST ACC	PRE	RECALL	F1
-5	30	4505	10513	1.78E-06	1	0.1	0.975	0.95	0.93	0.95	0.93
-5	25	3754	11254	1.32E-06	1	0.23	0.95	0.9	0.9	0.91	0.88
-5	20	3003	12015	1.37E-06	1	0.13	0.95	0.91	0.9	0.92	0.899
-5	15	2252	12766	1.69E-06	1	0.09	0.975	0.937	0.925	0.944	0.924
-5	10	1501	13517	1.03E-06	1	0.12	0.975	0.92	0.91	0.93	0.91
-5	5	750	14268	1.87E-06	1	0.03	0.9875	0.955	0.94	0.96	0.94

Table 6.4 Training and Testing Report for the Learning Rate of 0.00001 (main model)

LEARNING RATE	TEST SIZE	NO OF TEST	NO OF TRAIN	TRAIN LOSS	TRAIN ACC	TEST LOSS	VALID ACC	TEST ACC	PRE	RECALL	F1
-4	30	4505	10513	9.02E-07	1	0.05	0.9688	0.979	0.967	0.98	0.97
-4	25	3754	11254	1.15E-07	1	0.03	0.9937	0.89	0.898	0.9	0.877
-4	20	3003	12015	1.25E-07	1	0.06	0.975	0.96	0.94	0.96	0.95
-4	15	2252	12766	1.15E-04	1	0.09	0.956	0.97	0.957	0.975	0.96
-4	10	1501	13517	1.13E-05	1	0.05	0.9875	0.94	0.93	0.95	0.93
-4	5	750	14268	2.27E-05	1	0.06	0.9688	0.976	0.96	0.97	0.968

The number of training images ranges from 10513 to 14268 depending on the train-test split and the total number of images is 15018 (since images for multiple objects are added). The highest validation accuracy achieved is 0.9937 for 75/25 and 0.0001, but it doesn't perform as well for the test set. The training report for the top 3 accurate models is presented below.

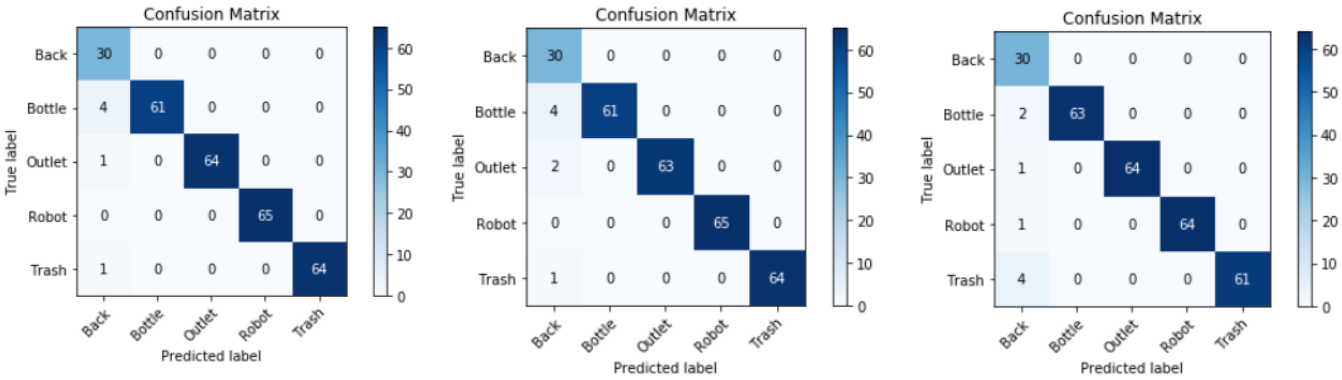


70/30 (0.0001) 95/5 (0.0001) 85/15 (0.0001)

70/30 (0.0001) 95/5 (0.0001) 85/15 (0.0001)

Figure 6.3 Training Summary for Top 3 Multi-Class Classification Models

The number of epoch for each training is 1000 and all the training in the above figure did not use all the epochs. The first one only used about half of the epoch, since the validation loss stops decreasing and the training is stopped by an early stop with patience of 10. All the top accurate models are the ones that are trained with a 0.0001 learning rate. The model is evaluated using the test set of 290 images containing both one object and multiple images. The following confusion matrix represents the result of the evaluation for the top 3 accurate models.



70/30 (0.0001) 95/5 (0.0001) 85/15 (0.0001)

Figure 6.4 Test Confusion Matrix for top 3 Multi-Class Classification Models

The left confusion matrix is for the model trained with a 70/30 train-test split and 0.0001 learning rate, and it is the most accurate model with a value of 0.979 (98%). The other 2 have a number of misclassification with an accuracy of 0.976 and 0.97.

Table 6.5 Model Comparison With Related Works

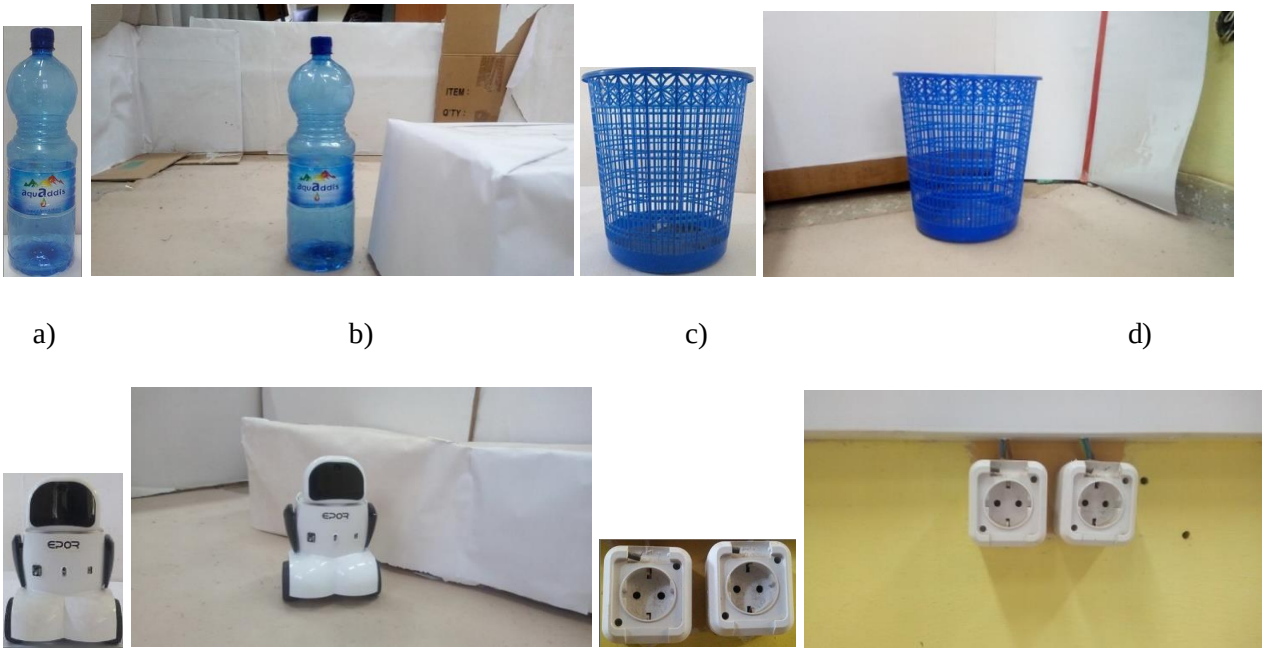
	Accuracy
SURF+SVM [1]	85%
Geometric features + SVM [8]	73.1%
CNN pipeline [7]	84.2%
SIFT + matching [35]	95%
Color Histogram + SIFT [6]	86%
Proposed Model	98%

As shown in the above table the proposed model has the highest accuracy and is able to recognize multiple objects within the same image. Deep feature-based models proposed by other related works do not run in real-time and also have lower mAP.

Figure 6.5 Sample Multiple Object Recognition

6.2.2 Object Localization

Object localization is applied to the image using local features depending on the probability that an object exists in the image. The sample database image and scene image are given below.



a) b) c) d)

e) f) g) h)

Figure 6.6 Sample Database and Scene Images

Image a, c, e, and g are database images and b, d, f, and h are sample scene images for a water bottle, trash can, EPOR robot, and outlet respectively. To localize an object in an image after its existence is known by the object classification, different feature extraction algorithms are evaluated. The detected

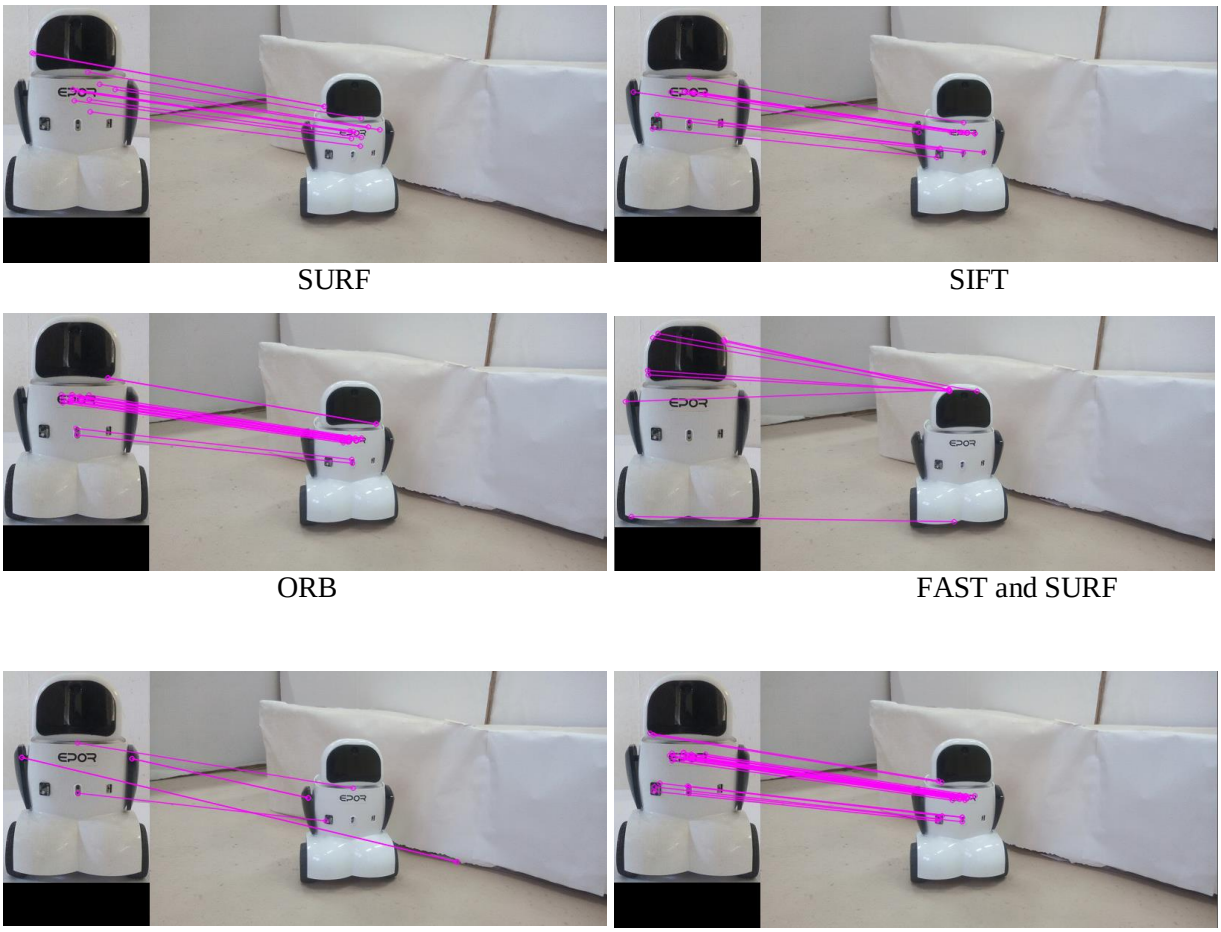
feature should be robust, efficient and fast enough that an object can be detected in different lighting, scale and viewpoint. SIFT, SURF, ORB, FAST+SURF, SIFT+SURF and ORB+SURF feature extraction algorithms are evaluated. In the different combinations of feature detectors and descriptors, SURF is chosen as a feature descriptor because of its accurate and fast feature representation [37]. The feature of an image (EPOR Robot is taken as a sample) detected and described by all chosen combination as shown in the bellow figure.

SURF SIFT ORB FAST and SURF

SIFT and SURF ORB and SURF

Figure 6.7 Feature Extraction for Sample EPOR Robot

SURF and FAST+SURF shown more false keypoint detection than others, where a plain surface is detected as a key-point. The number of key-points for both database and scene image by each feature extraction algorithm is recorded for comparison. Feature extraction time for scene image is also recorded since the feature of database image is extracted once, it does not affect the algorithm speed. The detected and described a feature of database image and scene image is matched for evaluating the matching accuracy and speed.



SURF SIFT

ORB FAST and SURF

SIFT and SURF ORB and SURF

Figure 6.8 Sample Feature Matching Using FLANN

The number of matches and correct matches for each feature extractor is recorded and given in the bellow table. FAST+SURF and ORB have the lowest detection time for 358 and 394 key-points respectively since ORB also uses oriented FAST for detecting features. The scene image used for evaluation has a size of 615 x 346 x 3.

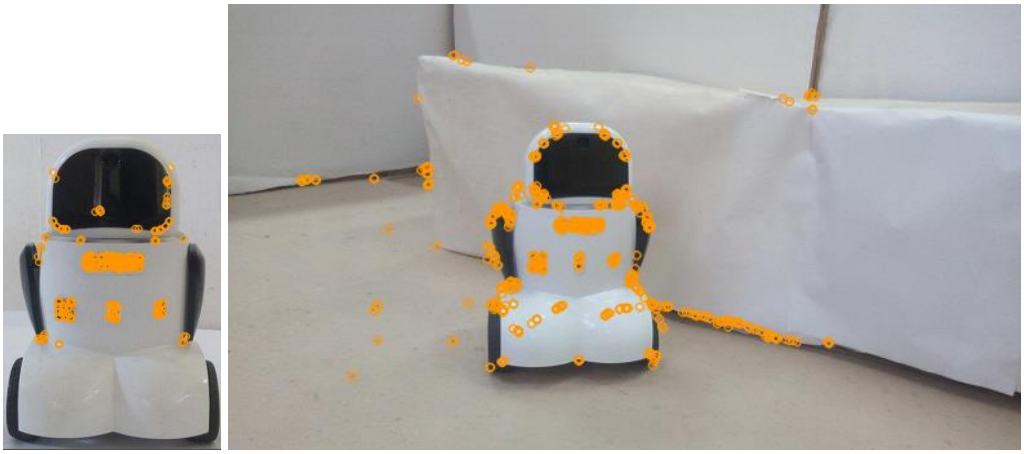
Table 6.6 Comparison Table for Each Feature Extractor

	Number of Key-point (database)	Number of Key-point (scene)	Detection Time (sec)	Number of Matches	Number of Correct matches	Correctness (%)	Matching Time (sec)
SURF	179	434	0.125s	179	33	18.4	0.016s
SIFT	81	96	0.109s	81	23	28.3	0.016s
ORB	265	394	0.016s	265	32	12	0.016s

FAST and SURF	293	358	0.015s	293	47	16	0.016s
SIFT and SURF	81	96	0.109s	81	10	12.3	0.016s
ORB and SURF	265	394	0.093s	265	37	14	0.016s

SIFT detects the least number of key-points and has the least number of correct matches compared to others. The detection time of SIRF and SIFT+SURF is also slower than others. ORB and ORB+SURF have the same number of key-points for both images but ORB+SURF has a greater number of correct matches. SIFT features are small and the most accurate but, slower than ORB+SURF. Even though FAST+SURF is the fastest and has a number of correct matches, its feature is not robust since it does not include orientation. ORB detects feature fast but its descriptor (BRIEF) is partially variant. The proposed ORB+SURF represents feature fast and accurately compared to others.

In the proposed object localization algorithms first, the feature is detected using the ORB feature detector and described using SURF.



a) b)

Figure 6.9 Feature Detection Sample for EPOR Robot Database Image and Sample Scene Image

The above figure shows the detected feature of the EPOR Robot image in the database (a) and scene (b). As shown in the image (a) the detected key-points are on the text and corners on the holes. Since ORB adds orientation to FAST in feature detection and these features are described using SURF, the feature is robust to scale, rotation, and illumination. The extracted features are matched using

FLANN to get the location of the object in the scene image. As shown in the bellow figure there are incorrect matches, since FLANN approximates the nearest neighbor.

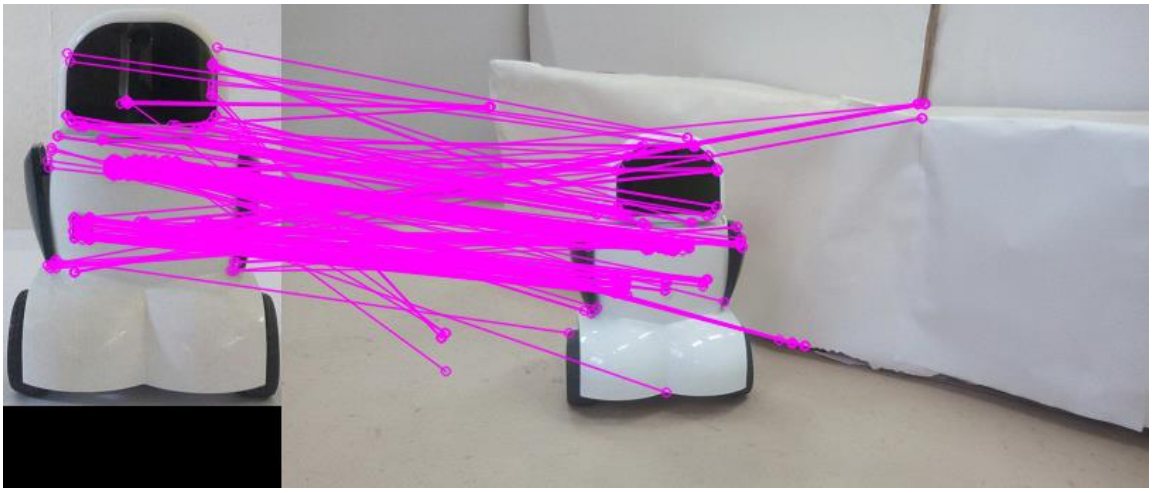


Figure 6.10 Feature Matching Sample Using ORB+SURF and FLANN

To remove the incorrect matches the Lowe ratio test with a threshold value of 0.85 is applied.

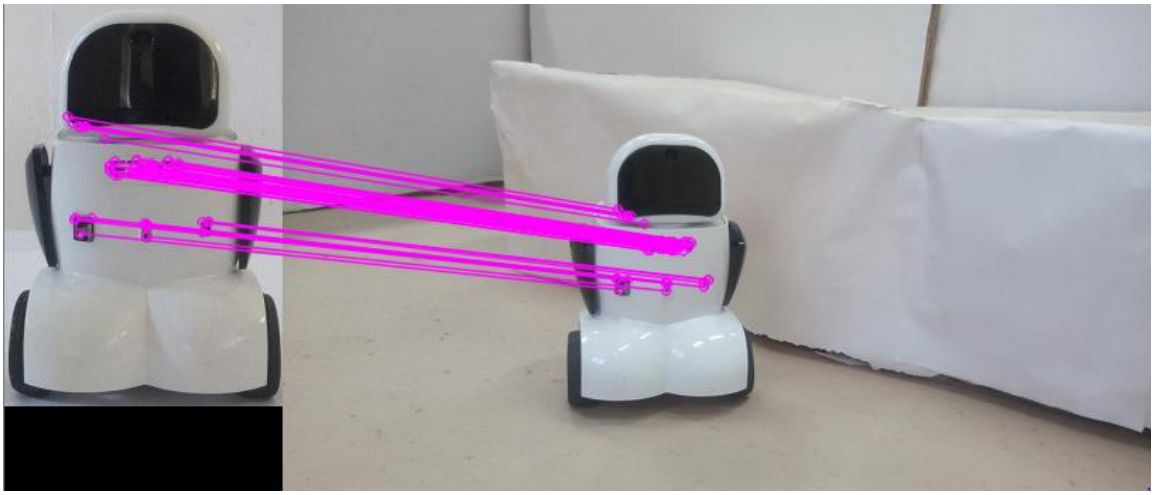


Figure 6.11 The Best Matches Above the Threshold for a Sample Image

After getting the best matches it is sorted based on its distance and the top 10 is selected. The location of each key-point is extracted from the select top matches, which is used to calculate the homography matrix and the mask for inliers. All the key-points in the mask are used to calculate the bounding box.

Figure 6.12 Bounding Box From Inliers for a Sample Image

The same techniques are applied for detecting multiple objects in the same image. For each object where the existence probability of that object in an image is higher than 0.6, object localization is applied. The following figure is shown some sample images for multiple object detection.

Figure 6.13 Sample Object Detection for Multiple Objects.

6.3 Object Detection Integration with Mobile Robot Navigation

The object detection algorithm is integrated with ROS so that the Turtlebot can interpret the environment by detecting objects while navigating through the environment. The integration is done by adding nodes for detection in mapping and navigation stages.

6.3.1 Environment Mapping

The map of the environment and object is created by using SLAM and the object detection algorithm respectively. The mapping process is visualized using the ROS visualization tool (RViz) as shown in the following figure. The image in the left shows the two-dimensional Occupancy Grid Map (OGM) created at an early stage and the green line represents the scan information from the LiDAR. The white represents the area that the robot can move and the black represents the occupied area.

The position and orientation are used to localize the robot on the map. The right image represents the output of the object detection algorithm, which is visualized using RQT. The image shows when the EPOR Robot is detected by the object detection algorithm while the Turtlebot is moving around the environment. The object detection algorithm takes an input image from the Raspberry PI camera mounted on the top Turtlebot3 and detects all the objects in the image and publishes the image with bounding boxes, class name, probability and distance to the robot.

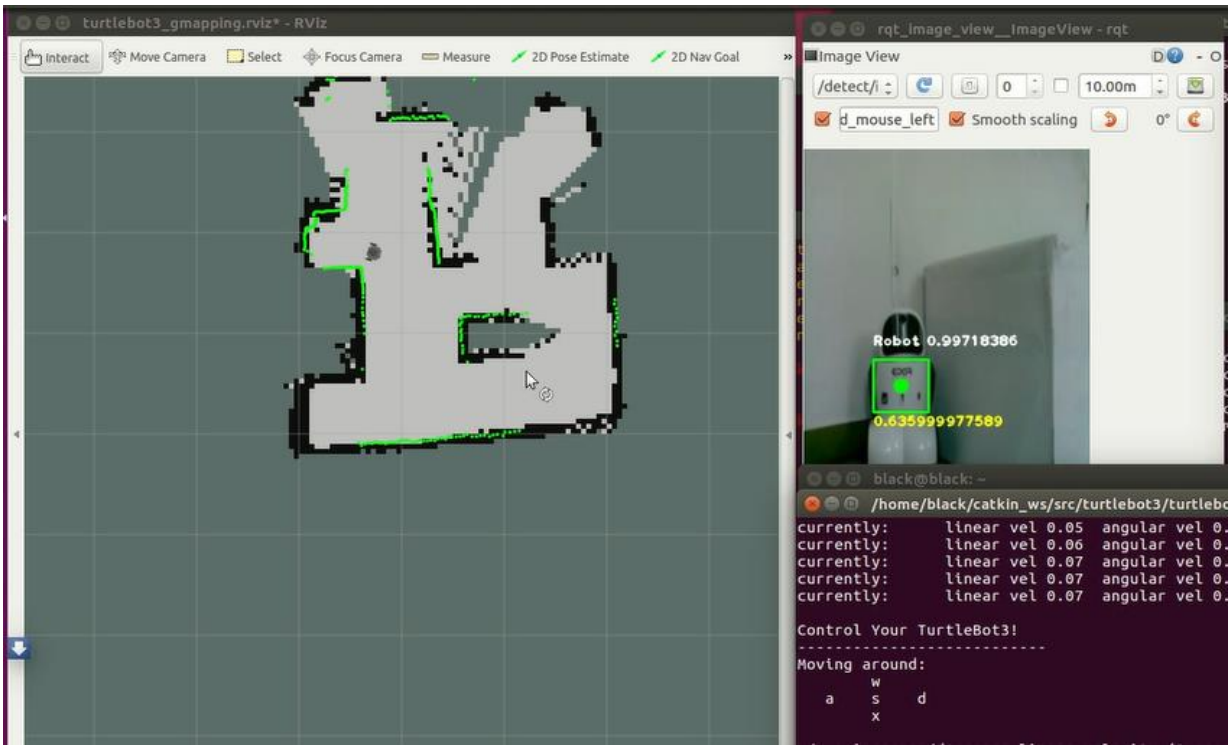


Figure 6.14 The Mapping Process and Object Detection Sample Visualized by RViz and RQT

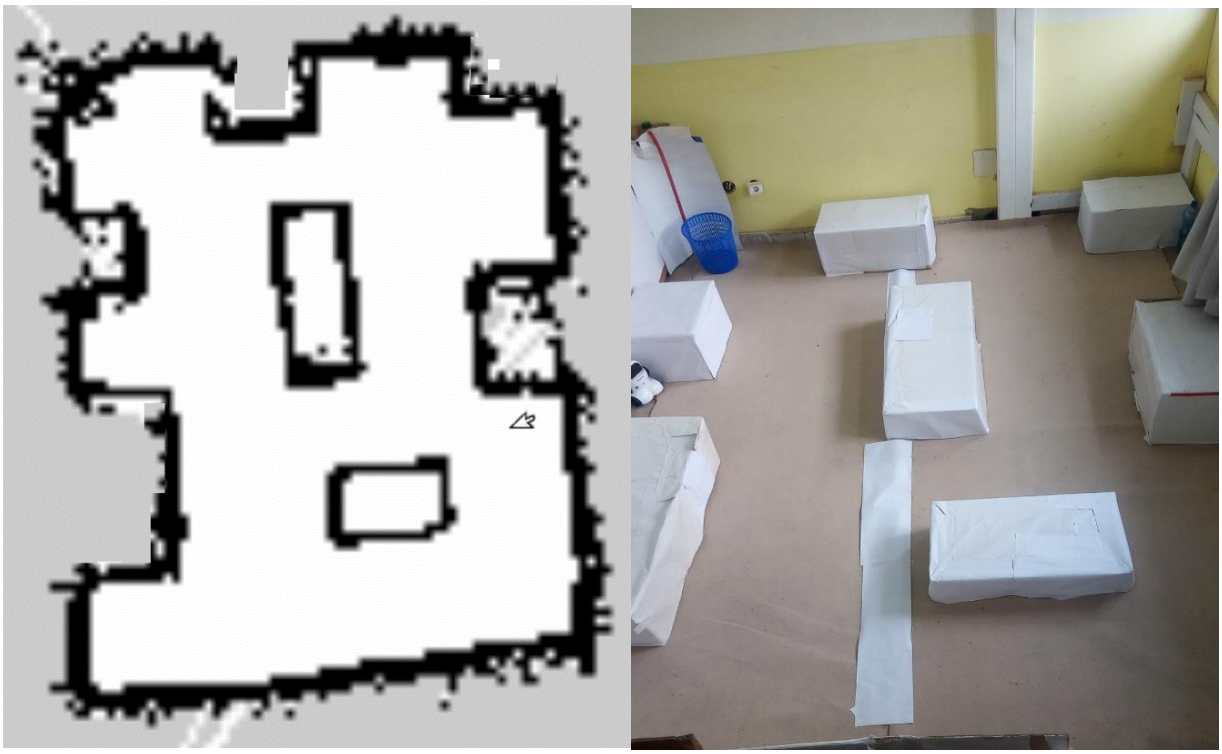


Figure 6.15 The Map Created for the Test-bed

The above figure shows the created map and the testbed. When the object detection algorithm detects an object it saves the position and orientation of the Turtlebot. These positions and orientations represent the pose of the robot for detecting that object. The location from where the robot can detect each object in the test-bed can be visualized in the following figure.

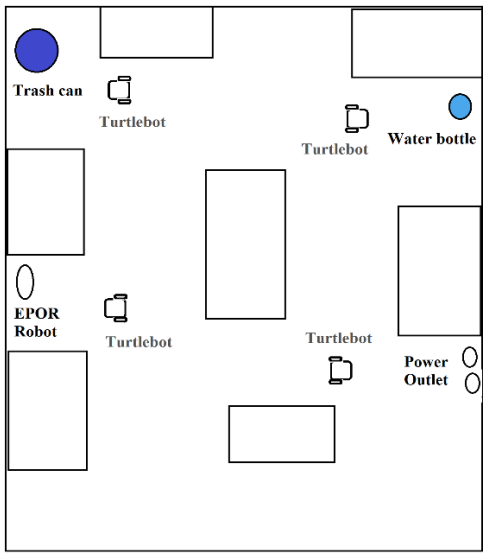


Figure 6.16 Sample Object Map for Testbed

6.3.2 Navigation Using Object Detection Algorithm

Using the created base environment and object map in the mapping step, the Turtlebot can navigate the environment from some point to another. The user gives a command to the robot using text (the name of the object in the given implementation scenario) through a command node. This node takes this command and finds its location from the object map. It creates a goal and executes using the location of the given command. Using the following scenarios the performance of the object detection and navigation is evaluated.

The first scenario is to navigate the environment from the starting pose to EPOR Robot. This scenario can be visualized in the real service robot scenario where the human (user) wants the robot (Turtlebot) to bring the EPOR Robot. First, the Turtlebot should know where the EPOR Robot is and should be able to navigate to that location and look for the Robot. The object detection algorithm is used to detect the EPOR Robot at some location when building a map or in some other scenario (like navigating for some other reason like how the human mind works) and then remember that location for the future. When the robot is given a task to find that object, it just has to navigate to that location and start looking for it again using the object detection algorithm.

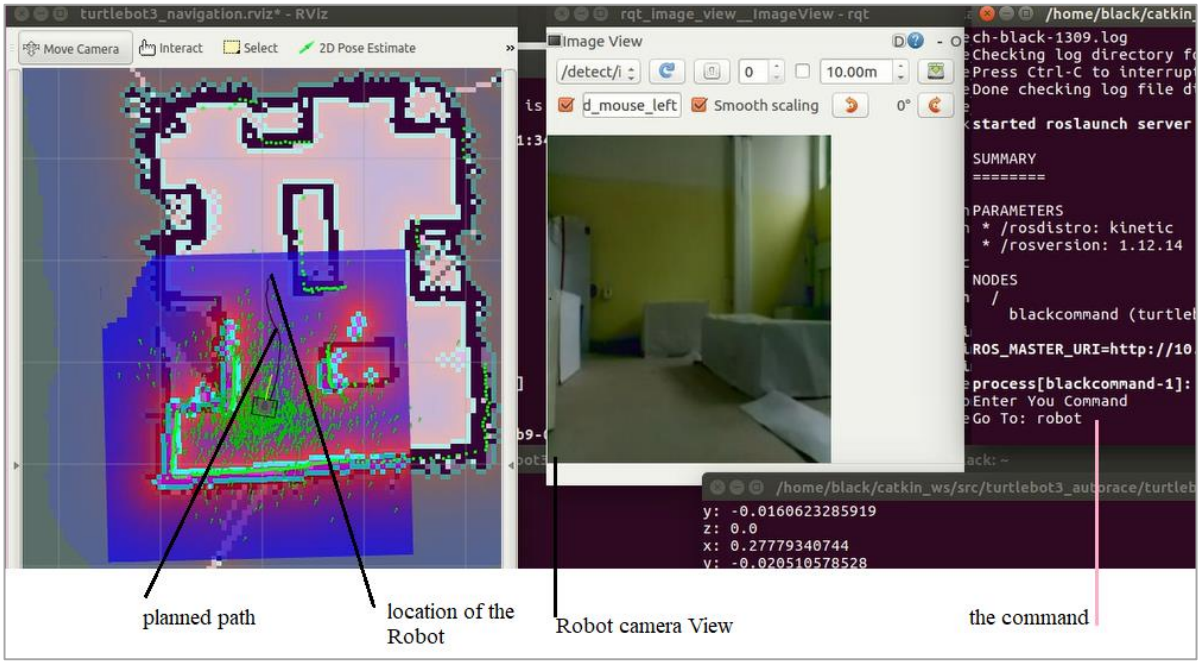


Figure 6.17 First Navigation Scenario Visualized by RViz and RQT

When the command is executed the path is planned by the global path planner to the object location as visualize in the above image by a blue line. It detects the EPOR Robot after it reaches the position as shown in the following figure.

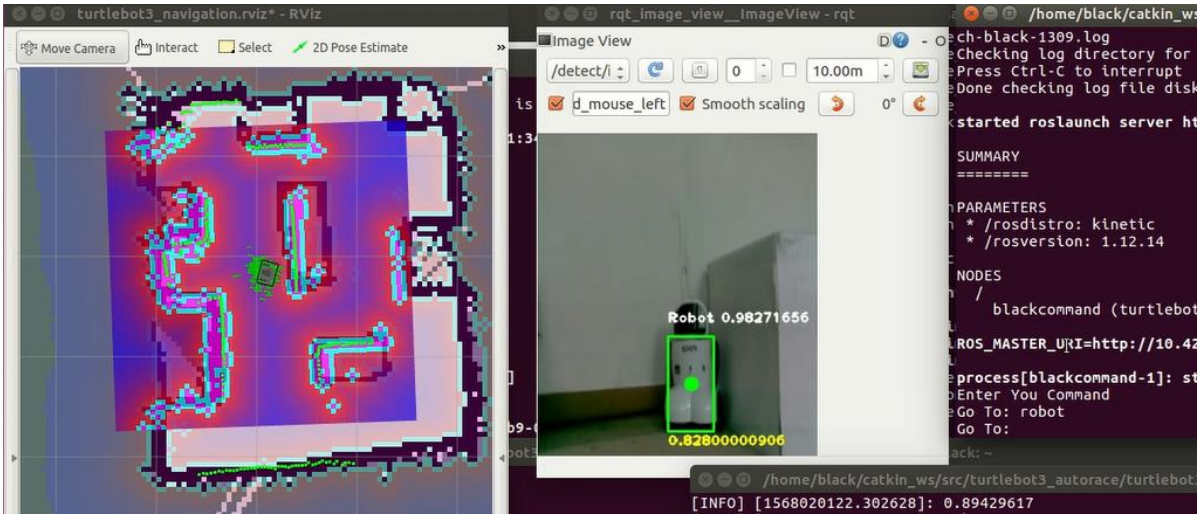


Figure 6.18 Object Detection Sample for the First Navigation Scenario

The object is bounded by a green box and the yellow text in the bottom of the bounding box is the distance of the robot to the object. The object is detected from 0.83 meters and the algorithm is 98% sure that the object is an EPOR robot. The second scenario is navigating from the starting point to the location of the bottle. The following figures show the navigation path and how the object is detected once the location is reached.

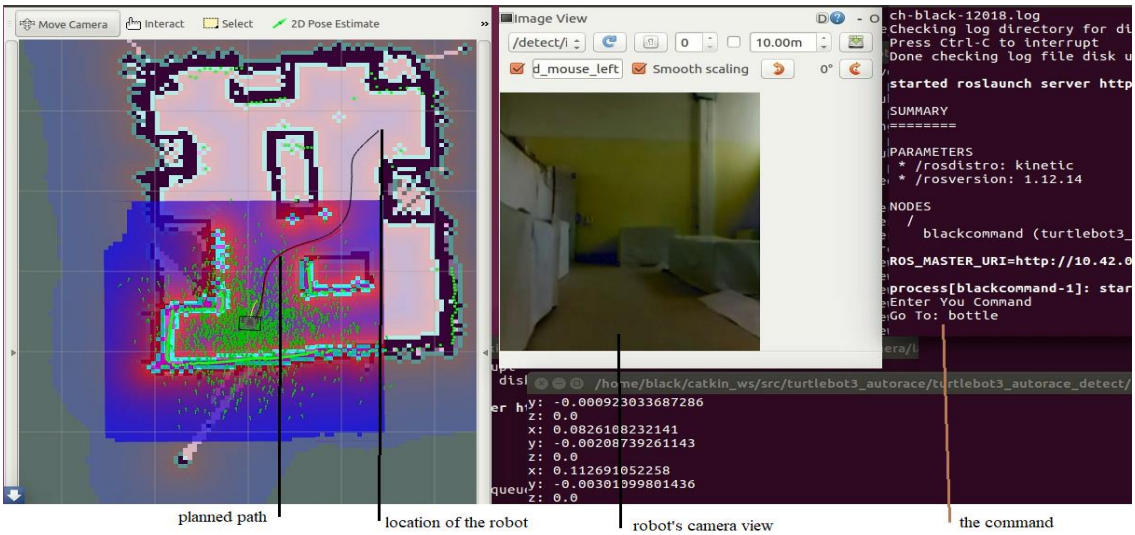


Figure 6.19 Second Navigation Scenario Visualized by RViz and RQT

Figure 6.20 Object Detection Sample for the Second Navigation Scenario

When it reaches the location the Turtlebot starts to look for the object. The object is detected at a distance of 0.67 meters and the algorithm is 95% sure that the object is a bottle. After detection, the object a certain action like picking and manipulating can be taken. The algorithm can process one image in 0.075 seconds so it can process 14 frames per second on the desktop computer.

Putting the object recognition, object localization, and navigation, what kind of object exists, in what part of the image and at what distance does the object is located from the robot can be acquired. The following are some of the sample images by combining them all together.

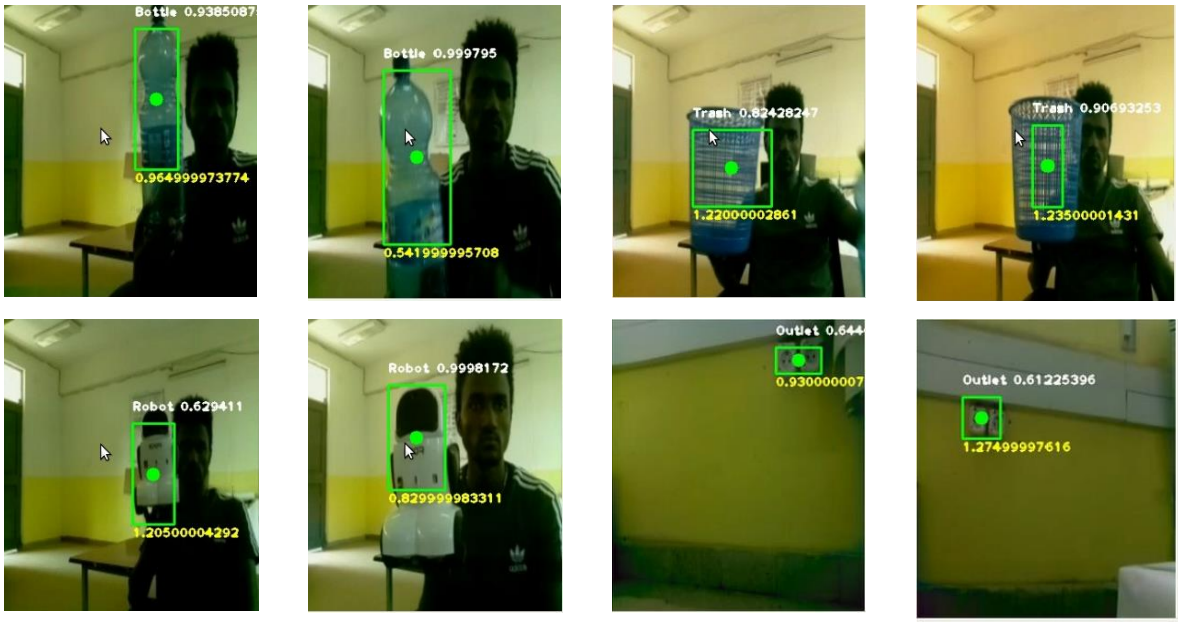


Figure 6.21 Object Detection Sample by Combining All Together

CHAPTER SEVEN

7 Conclusion and Future Work

7.1 Conclusion

To make mobile service robots truly ubiquitous to complex and dynamic indoor environments, they should be able to understand the surrounding environment beyond the capability of avoiding obstacles, navigating autonomously, and building maps. It's necessary to build a reliable and fast detection system to enhance the performance of indoor robot navigation. Mobile robots do not come with heavy computing power so the detection algorithm response time should be sufficient enough that the robots can make a decision fairly quickly. Multiple object recognition and localization for mobile robot navigation in a dynamic indoor environment is presented in this work.

The proposed object detection algorithm is a combination of two steps. The first one is recognition, which is responsible for providing the probability of each class for a given image. For recognition of the image, the CNN-based deep feature extraction technique is applied with a sigmoid classification layer. The model is trained with 15,018 images collected from the computer vision lab and testbed. The model is trained with different parameters in two steps, one for a single object and the other one for multiple objects. The best performing model (100% accurate for the given test set) in the first step is used as a feature extractor for the second one. 98% accuracy (97% mAP) is achieved for multiple object recognition. The output of the recognition step is used to localize the object in the image, so the second step is responsible for localizing the object based on the probability provided by the recognition step.

For localization different combination of Feature detectors and descriptors are compared with the proposed ORB+SURF feature extractor to evaluate its performance. The proposed ORB+SURF represents features fast and accurately compared to others. The feature of both database and scene images are detected by ORB and described by SURF and then matched using FLANN. The matches are tested (using threshold), sorted and the tops are selected to calculate the bounding box for localization. If the probability generated by the recognition step for each object in an image is above 0.6 then the bounding box is generated for it.

The object detection algorithm is integrated with ROS on top of Turtlebot3 Burger to evaluate its performance. The object detection algorithm takes an input image from the Raspberry PI camera mounted on the top Turtlebot3 and detects all the objects in the image and publishes the image with bounding boxes, class name, probability and the distance of the object to the robot. First, the base map of the environment is created using SLAM while simultaneously detecting an object in the environment and saving the pose of the robot. Then different navigation experiments are performed by adding 2 nodes to the ROS, one for navigation object detection and the other for taking the user's navigation command. The navigation experiments are executed successfully, where the Turtlebot is able to navigate to the destination and detect the desired object.

The experimental results indicate that the developed object detection algorithm has better performance than the related works. Its accuracy, detection speed (real-time) and feasibility makes it a better choice for mobile robots that have limited resource to perform detection. When it is integrated with the ROS and Robot it also provides distance information from the scan information of LiDAR, which has less computing expenses.

7.2 Future Work

The proposed deep-learning-based recognition model performs well for both single objects and multiple objects, but the localization algorithm can be modified for the localization of multiple objects and multiple instances of an object. Since the descriptor of the scene image is matched with the database image for localization the number of instances is unknown. The descriptor of the scene image can be clustered using algorithms like Mean-shift so that the instance can be matched with each cluster.

The object detection algorithm can be integrated into more advanced and high-level navigations like semantic navigation for human-level navigation. The relationship between objects can be used for navigating from one point to another.

Reference

[1] M. Takacs, T. Bencze, M. Z. Szabo-Resch, and Z. Vamossy, "Object recognition to support indoor robot navigation," *CINTI 2015 - 16th IEEE Int. Symp. Comput. Intell. Informatics, Proc.*, pp. 239–242, 2016.

[2] C. Astua, R. Barber, J. Crespo, and A. Jardon, "Object detection techniques applied on mobile robot semantic navigation," *Sensors (Switzerland)*, vol. 14, no. 4, pp. 6734–6757, 2014.

[3] W. H. Organization, "GLOBAL DATA ON VISUAL IMPAIRMENTS," 2010.

[4] D. H. A. and Z. Y. B. Fashe Markos Cherinet, Sophia Yoseph Tekalign, "Prevalence and associated factors of low vision and blindness among patients attending St. Paul's Hospital Millennium Medical College, Addis Ababa, Ethiopia," 2018.

[5] H. Jabnoun, F. Benzarti, and H. Amiri, "Visual substitution system for blind people based on SIFT description," *6th Int. Conf. Soft Comput. Pattern Recognition, SoCPaR 2014*, pp. 300–305, 2015.

[6] T. R. and m. B. W. Obaid, "Real-Time Color Object Recognition and Navigation for QUARC QBOT2," *Int. Conf. Comput. Appl.*, 2017.

[7] X. Ding *et al.*, "Indoor object recognition using pre-trained convolutional neural network," *ICAC 2017 - 2017 23rd IEEE Int. Conf. Autom. Comput. Addressing Glob. Challenges through Autom. Comput.*, 2017.

[8] A. C. Hernández, C. Gómez, J. Crespo, and R. Barber, "Object Classification in Natural Environments for Mobile Robot Navigation," *Proc. - 2016 Int. Conf. Auton. Robot Syst. Compet. ICARSC 2016*, pp. 217–222, 2016.

[9] A. C. Hernandez, C. Gomez, J. Crespo, and R. Barber, "Adding uncertainty to an object detection system for mobile robots," *Proc. - 6th IEEE Int. Conf. Sp. Mission Challenges Inf. Technol. SMC-IT 2017*, vol. 2017-Decem, pp. 7–12, 2017.

[10] H. Lee *et al.*, "Object recognition for vision-based navigation in indoor environments without using image database," *Proc. Int. Symp. Consum. Electron. ISCE*, pp. 1–2,

2014.

[11] IROBOT Corporation, “iRobot: Vacuum, Mop, & Lawn Mower.” [Online]. Available: https://www.irobot.com/. [Accessed: 19-Sep-2019].

[12] Q. Y. J. J. Z. Han, “ONLINE FEATURE EVALUATION FOR OBJECT TRACKING USING KALMAN FILTER,” *IEEE Access*, 2008.

[13] C. G. S. A. S. Saravanakumar, A. Vadivel, “HUMAN OBJECT TRACKING IN VIDEO SEQUENCES,” vol. 2, no. 1, 2011.

[14] D. G. Viswanathan, “Features from Accelerated Segment Test (FAST) Deepak Geetha Viswanathan 1.”

[15] E. Senn, “BRIEF: Binary Robust Independent Elementary Features,” vol. 2011, no. 10/12, 2011.

[16] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “ORB: and efficient alternative to SIFT and SURF,” *2011 IEEE Int. Conf. Comput. Vis.*, 2011.

[17] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *Int. J. Comput. Vis.*, pp. 91–110, 2004.

[18] L. Van Gool, “SURF : Speeded up robust features SURF : Speeded Up Robust Features,” no. July 2006, 2016.

[19] H. Sun, L. Yan, P. Mooney, and R. Liang, “A new method for moving object detection using variable resolution bionic compound eyes,” *Int. J. Phys. Sci.*, vol. 6, no. 24, pp. 5618–5622, 2011.

[20] W.-L. L. ; J. J. Little, “Simultaneous Tracking and Action Recognition using the PCA-HOG Descriptor,” *3rd Can. Conf. Comput. Robot Vis.*, 2006.

[21] Y. Zhong, A. K. Jain, and M. P. Dubuisson-Jolly, “Object tracking using deformable templates,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 5, pp. 544–549, 2000.

[22] B. Alexe, T. Deselaers, and V. Ferrari, “Measuring the objectness of image windows,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 11, pp. 2189–2202, 2012.

[23] J. R. R. Uijlings, K. E. A. Van De Sande, T. Gevers, and A. W. M. Smeulders, "Selective search for object recognition," *Int. J. Comput. Vis.*, vol. 104, no. 2, pp. 154–171, 2013.

[24] M. M. Cheng, Y. Liu, W. Y. Lin, Z. Zhang, P. L. Rosin, and P. H. S. Torr, "BING: Binarized normed gradients for objectness estimation at 300fps," *Comput. Vis. Media*, vol. 5, no. 1, pp. 3–20, 2019.

[25] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient Graph-Based Image Segmentation," pp. 1–26, 2015.

[26] I. Endres and D. Hoiem, "Category-independent object proposals with diverse ranking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 2, pp. 222–234, 2014.

[27] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-Based Convolutional Networks for Accurate Object Detection and Segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 1, pp. 142–158, 2016.

[28] R. Girshick, "Fast R-CNN," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 Inter, pp. 1440–1448, 2015.

[29] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017.

[30] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016.

[31] W. Liu *et al.*, "SSD: Single shot multibox detector," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9905 LNCS, pp. 21–37, 2016.

[32] C. World, "YOLO9000:Better , stronger , faster," no. April, 2007.

[33] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 2018.

[34] M. R. B. S. Riisgaard, "SLAM for Dummies : A Tutorial Approach to Simultaneous

Localization and Mapping," vol. 80, no. 4, pp. 699–705, 2000.

[35] H. Jabnoun, F. Benzarti, and H. Amiri, "Object detection and identification for blind people in video scene," *Int. Conf. Intell. Syst. Des. Appl. ISDA*, vol. 2016-June, pp. 363–367, 2016.

[36] N. Diriba, "A HYBRID ALGORITHM FOR FAST DETECTION AND RECOGNITION OF SIGNAGE AND OBSTACLE AVOIDANCE FOR ROBOT NAVIGATION," 2019.

[37] R. Pedro, "Object recognition for a service robot," 2015.

[38] Yeong-Hwa Chang ; Ping-Lun Chung ; Hung-Wei Lin, "Deep learning for object identification in ROS-based mobile robots," 2018.

[39] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64270–64277, 2018.

[40] B. Istanbul, "Analysis of Feature Detector and Descriptor Combinations with a Localization Experiment for Various Performance Metrics Ertugrul BAYRAKTAR* , Pınar BOYRAZ."

[41] E. M. Solution, "Edraw Max : Professional Diagram Solution," *EDraw Max*, 2019. [Online]. Available: http://www.edrawmax.com/EDrawMax.php. [Accessed: 02-Feb-2019].

[42] Tensorflow, "An open source machine learning framework for everyone." [Online]. Available: https://www.tensorflow.org/.

[43] J. Hale, "Deep Learning Framework Power Scores," 2018. [Online]. Available: https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a. [Accessed: 21-May-2019].

[44] J. Hale, "deep-learning-framework-power-scores-2018," *"Kaggle," towardsdatascience.com*. [Online]. Available: https://www.kaggle.com/discdiver/deep-learning-framework-power-scores-2018? [Accessed: 13-May-2019].

[45] F. C. K. Group, “Keras documentation,” 2015. [Online]. Available: https://keras.io/. [Accessed: 12-Nov-2018].

[46] OpenCV.org, “Opencv.” [Online]. Available: https://opencv.org/about.html. [Accessed: 27-Oct-2018].

[47] H. I. El-Zorkany, *Robot Programming.*, vol. 23, no. 4. 1984.

[48] R. Fundation, “Ros gui Development tool documentation,” 2019. [Online]. Available: http://www.wiki.ros.org/rqt.

[49] Robotis, “Turtlebot 3,” 2018. [Online]. Available: http://www.robotis.us/turtlebot-3/. [Accessed: 21-Oct-2018].

[50] ROBOTIS, “Turtlebot3 emanual,” 2018. [Online]. Available: http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/. [Accessed: 01-Aug-2019].

[51] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 4510–4520, 2018.

[52] A. G. Howard *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” 2017.

[53] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 770–778, 2016.

[54] D. L. Muja, Marius, “FLANN - Fast Library for Approximate Nearest Neighbors,” 2013.

Appendixes

Appendix A: Sample Code for Training the Model

```
def load_data(dataset):
    print(dataset+"----------")
    datas = []
    data = []
    labels = []
    l = -1
    alldirnames = []
    k = 0

    for (dirpath, dirnames, filenames) in walk(dataset):
        datas.extend(filenames)
        if k == 0:
            alldirnames.extend(dirnames)
        k = k + 1
    u = 0
    print(str(len(alldirnames))+" Classes")

    for i in range(len(alldirnames)):
        for filename in glob.glob(dataset+"/"+alldirnames[i] +'/*.jpg'):

            data.append(filename)
            hot_shot = [0,0,0,0,0]
            hot_shot[i] = 1
            labels.append(hot_shot)
            u = u + 1

        print(alldirnames[i]+" :    "+str(u))
        u = 0

    return data, labels

def myBatchGenerator(batch_size, images, labels):

    batch_image = []
    batch_label = []

    batch_counter = 0
    while True:
        while batch_counter < batch_size:
```

```
            rand_ind = np.random.randint(len(images))

            filename = images[rand_ind]
            image_string = tf.read_file(filename)

            image = cv.imread (filename)
            image = cv.resize(image,(224,224))
            batch_image.append (image)


            batch_label.append(labels[rand_ind])

            batch_counter = batch_counter + 1
        BI = np.array(batch_image)
        BL = np.array(batch_label)
        yield BI, BL

def load_data(dataset):
    print(dataset+"----------")
    datas = []
    data = []
    labels = []
    l = -1
    alldirnames = []
    k = 0

    for (dirpath, dirnames, filenames) in walk(dataset):
        datas.extend(filenames)
        if k == 0:
            alldirnames.extend(dirnames)
        k = k + 1
    u = 0
    print(str(len(alldirnames))+" Classes")
    for i in range(len(alldirnames)):
        for filename in glob.glob(dataset+"/"+alldirnames[i] +'/*.jpg'):

            data.append(filename)
            hot_shot = [0,0,0,0,0]
            if(i < 5):
                hot_shot[i] = 1
            elif(i == 5):
                # bottle and outlet
                hot_shot = [0,1,1,0,0]
```

```
            elif(i == 6):
                # bottle and trash
                hot_shot = [0,1,0,0,1]
            elif(i == 7):
                #outlet and trash
                hot_shot = [0,0,1,0,1]
            elif(i == 8):
                # bottle and robot
                hot_shot = [0,1,0,1,0]
            elif(i == 9):
                # outlet and robot
                hot_shot = [0,0,1,1,0]

            elif(i == 10):
                # robot and trash
                hot_shot = [0,0,0,1,1]
            elif(i == 11):
                # bottle, outlet and robot
                hot_shot = [0,1,1,1,0]
            elif(i == 12):
                # bottle, outlet and trash
                hot_shot = [0,1,1,0,1]
            elif(i == 13):
                # bottle, robot and trash
                hot_shot = [0,1,0,1,1]
            elif(i == 14):
                # outlet, robot and trash
                hot_shot = [0,0,1,1,1]
            elif(i == 15):
                # bottle, outlet, robot and trash
                hot_shot = [0,1,1,1,1]


            labels.append(hot_shot)
            u = u + 1

        print(alldirnames[i]+" :    "+str(u))
        u = 0

    return data, labels
```

Appendix B: Sample code for object detection in Mapping Node

```
class BlackDetector():
    def __init__(self):
        self.fnPreproc()


        self.sub_image_type = "raw" # you can choose image type "compressed", "raw"
        self.pub_image_type = "compressed" # you can choose image type "compressed", "raw"

        if self.sub_image_type == "compressed":
            # subscribes compressed image
            self.sub_image_original = rospy.Subscriber('/detect/image_input/compressed',
             CompressedImage, self.cbDetectObject, queue_size = 1)
        elif self.sub_image_type == "raw":
            # subscribes raw image
            self.sub_image_original = rospy.Subscriber('/detect/image_input', Image,
            self.cbDetectObject, queue_size = 1)

        self.pub_Object_detection = rospy.Publisher('/detect/image_blackmap', UInt8, queue_size=1)

        self.odem_sub = rospy.Subscriber('/odom', Odometry, self.theOdem)
        self.dist_sub = rospy.Subscriber("/scan", LaserScan, self.theDist)

        if self.pub_image_type == "compressed":

            self.pub_image_Object_detection = rospy.Publisher('/detect/image_output/compressed',
            CompressedImage, queue_size = 1)
        elif self.pub_image_type == "raw":

            self.pub_image_Object_detection = rospy.Publisher('/detect/image_output',
            Image, queue_size = 1)
```

```
def fnPreproc(self):
    # Initiate SIFT detector
    self.surf = cv2.xfeatures2d.SURF_create()
    self.detector = cv2.ORB_create( nfeatures = 1000 )

    self.position = []
    self.orientation = []
    self.distance = 0

    self.bottlePo = []
    self.trashPo = []
    self.outletPo = []
    self.robotPo = []

    self.dir_path = os.path.dirname(os.path.realpath(__file__))
    self.dir_path = self.dir_path.replace('turtlebot3_autorace_detect/nodes',
    'turtlebot3_autorace_detect/')
    self.dir_path += 'file/black/'
```

```
    img2 = cv2.imread(self.dir_path + 'bottle5.jpg')      # Database Image For : Bottle
    img3 = cv2.imread(self.dir_path + 'outlet.jpg')      # Database Image For : Outlet
    img4 = cv2.imread(self.dir_path + 'robot2.jpg')      # Database Image For : Robot
    img5 = cv2.imread(self.dir_path + 'trash.jpg')      # Database Image For : Trash

    self.dataImages = [img2, img3, img4, img5]

    kp2 = self.detector.detect(self.dataImages[0],None)
    kp3 = self.detector.detect(self.dataImages[1],None)
    kp4 = self.detector.detect(self.dataImages[2],None)
    kp5 = self.detector.detect(self.dataImages[3],None)

    kp, des2 = self.surf.compute(self.dataImages[0], kp2)
    kp, des3 = self.surf.compute(self.dataImages[1], kp3)
    kp, des4 = self.surf.compute(self.dataImages[2], kp4)
    kp, des5 = self.surf.compute(self.dataImages[3], kp5)

    self.keyp = [kp2, kp3, kp4, kp5]
    self.desc = [des2, des3, des4, des5]
```

```
    FLANN_INDEX_KDTREE = 0
    index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
    search_params = dict(checks = 50)

    self.flann = cv2.FlannBasedMatcher(index_params, search_params)

    self.classes = ["Back","Bottle", "Outlet", "Robot", "Trash"]

    #self.sess = tf.Session()
    self.graph = tf.get_default_graph()
    self.model = keras.models.load_model(self.dir_path+'blackcool.h5')

def theDist(self, msg):
    self.distance = msg.ranges[353]
    #print(msg.ranges[353])

def fnCalcMSE(self, arr1, arr2):
        squared_diff = (arr1 - arr2) ** 2
        sum = np.sum(squared_diff)
        num_all = arr1.shape[0] * arr1.shape[1] #cv_image_input and 2 should have same shape
        err = sum / num_all
        return err

def theOdem(self, msg):

    self.position = msg.pose.pose.position
    self.orientation = msg.pose.pose.orientation


max1 = 0
output = orig
with self.graph.as_default():
    #set_session(self.sess)
        pred = self.model.predict(np.array(images))
        out = pred[0]
        maxz = heapq.nlargest(5, range(len(out)), out.take)

        print(self.position)

        for i in range(5):

            max1 = maxz[i]

            if(out[max1]>0.6 and max1 != 0):
                rospy.loginfo(out[max1])
                rospy.loginfo(self.classes[max1])
```

```
            #find the keypoints ORB and descriptors SURF
            kp1 = self.detector.detect(img1,None)
            fdf, des1 = self.surf.compute(img1, kp1)

            kp2 = self.keyp[max1-1]
            dec2 = self.desc[max1-1]
            img2 = self.dataImages[max1-1]

            if(des1 is not None and dec2 is not None):
                if(len(des1) > 2 and len(dec2) > 2):
```

```
theMatch = self.flann.knnMatch(dec2, des1, k=2)
theMatch = sorted(theMatch, key = lambda x:x[0].distance)

goodF = [m1 for (m1, m2) in theMatch if m1.distance < 0.9 * m2.distance]

if len(goodF)>MIN_MATCH_COUNT:

    goodF = goodF[:10]

    src_pts = np.float32([ kp2[m.queryIdx].pt for m in goodF ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp1[m.trainIdx].pt for m in goodF ]).reshape(-1,1,2)

    M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,3.0)

    if M is not None:
        matchesMask = mask.ravel().tolist()
        mse = self.fnCalcMSE(src_pts, dst_pts)
        if mse < MIN_MSE_DECISION:

            h,w = img2.shape
```

```
    if(max1 == 1):
        self.bottlePo = [self.position.x, self.position.y, self.position.z,
        self.orientation.x, self.orientation.y, self.orientation.z, self.orientation.w]
    elif(max1 == 2):
        self.outletPo = [self.position.x, self.position.y, self.position.z,
        self.orientation.x, self.orientation.y, self.orientation.z, self.orientation.w]
    elif(max1 == 3):
        self.robotPo = [self.position.x, self.position.y, self.position.z,
        self.orientation.x, self.orientation.y, self.orientation.z, self.orientation.w]
```

```
    elif(max1 == 4):
        self.trashPo = [self.position.x, self.position.y, self.position.z,
        self.orientation.x, self.orientation.y, self.orientation.z, self.orientation.w]

    all_ob = []
    all_ob.append(self.bottlePo)
    all_ob.append(self.outletPo)
    all_ob.append(self.robotPo)
    all_ob.append(self.trashPo)


    semanticMap = open(self.dir_path + "sematicmap.txt", 'w')
    json.dump(all_ob, semanticMap)


    pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1, 1, 2)
    dst = cv2.perspectiveTransform(pts,M)
    #dst += (w, 0)
```

```
    o = 0
    allx = []
    ally = []
    for m in goodF:
        if(matchesMask[o] == 1):
            allx.append(kp1[m.trainIdx].pt[0])
            ally.append(kp1[m.trainIdx].pt[1])
        o = o + 1
        print(allx)
    minx = min(allx)-10
    miny = min(ally)-10
    maxx = max(allx)+10
    maxy = max(ally)+10

    xcenter = (maxx + minx) / 2
    ycenter = (maxy + miny) / 2


    output = cv2.rectangle(output,(int(minx), int(miny)),
    (int(maxx),int(maxy)),(0,255,0),2)
```

```
        output = cv2.circle(output,(int(xcenter), int(ycenter)),
        8, (0,255,0), -1)

        cv2.putText(output, self.classes[max1]+" "+str(out[max1]),
        (int(minx), int(miny)-15), 1, 1, (255,255,255), 2, 1)
        cv2.putText(output, str(self.distance), (int(minx),
        int(maxy)+15), 1, 1, (0,255,255), 2, 1)
```

Appendix C: Sample Code for Object Detection in Navigation Node

```
def updateObjectMap(self, max1):

    dir_path = os.path.dirname(os.path.realpath(__file__))
    dir_path = dir_path.replace('turtlebot3_autorace_detect/nodes',
    'turtlebot3_autorace_detect/')
    dir_path += 'file/black/'

    sf = open(dir_path + "sematicmap.txt")

    sematicMap = json.load(sf)

    if(max1 == 1):

        bottle = [self.position.x, self.position.y, self.position.z,
        self.orientation.x, self.orientation.y, self.orientation.z, self.orientation.w]
        outlet = sematicMap[1]
        robot = sematicMap[2]
        trash = sematicMap[3]

    elif(max1 == 2):
        outlet = [self.position.x, self.position.y, self.position.z,
        self.orientation.x, self.orientation.y, self.orientation.z, self.orientation.w]
        bottle = sematicMap[0]
        robot = sematicMap[2]
        trash = sematicMap[3]

    elif(max1 == 3):
        robot = [self.position.x, self.position.y, self.position.z,
        self.orientation.x, self.orientation.y, self.orientation.z, self.orientation.w]
        bottle = sematicMap[0]
        outlet = sematicMap[1]
        trash = sematicMap[3]
```

```
    elif(max1 == 4):
        trash = [self.position.x, self.position.y, self.position.z,
         self.orientation.x, self.orientation.y, self.orientation.z, self.orientation.w]
        bottle = sematicMap[0]
        outlet = sematicMap[1]
        robot = sematicMap[2]

    all_ob = []
    all_ob.append(bottle)
    all_ob.append(outlet)
    all_ob.append(robot)
    all_ob.append(trash)

    return all_ob
```

Appendix C: Sample Code for Command Node

```
# Author: Anteneh tilaye

import os
import json
import speech_recognition as sr
import rospy
import actionlib
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
from tf.transformations import euler_from_quaternion

def blackCommander():

    # read all objects location

    dir_path = os.path.dirname(os.path.realpath(__file__))
    dir_path = dir_path.replace('turtlebot3_autorace_detect/nodes', 'turtlebot3_autorace_detect/')
    dir_path += 'file/black/'

    sf = open(dir_path + "sematicmap.txt")

    sematicMap = json.load(sf)

    bottle = sematicMap[0]
    outlet = sematicMap[1]
    robot = sematicMap[2]
    trash = sematicMap[3]
```

```
    print("Enter You Command")
    # Accept a command
    while True:

        command = raw_input("Go To: ")

        if (str(command) == "bottle"):
            goTo(bottle)
        elif (str(command) == "outlet"):
            goTo(outlet)

        elif (str(command) == "robot"):
            goTo(robot)
        elif (str(command) == "trash"):
            goTo(trash)
        else:
            print("I Don't Understand")


def goTo(place):

    client = actionlib.SimpleActionClient('move_base',MoveBaseAction)
    client.wait_for_server()

    goal = MoveBaseGoal()
    goal.target_pose.header.frame_id = "map"
    goal.target_pose.header.stamp = rospy.Time.now()
    goal.target_pose.pose.position.x = place[0]
    goal.target_pose.pose.position.y = place[1]
    goal.target_pose.pose.position.z = place[2]

    #orlist = [place[3], place[4], place[5], place[6]]

    #(roll, pitch, yaw) = euler_from_quaternion(orlist)

    goal.target_pose.pose.orientation.x = place[3]
    goal.target_pose.pose.orientation.y = place[4]
    goal.target_pose.pose.orientation.z = place[5]
    goal.target_pose.pose.orientation.w = place[6] - 0.2
```

```
    client.send_goal(goal)
    wait = client.wait_for_result()
    if not wait:
        rospy.logerr("Action server not available!")
        rospy.signal_shutdown("Action server not available!")
    else:
        return client.get_result()

if __name__ == '__main__':
    try:
        rospy.init_node('blackcommand')
        result = blackCommander()
        if result:
            rospy.loginfo("Done")
    except rospy.ROSInterruptException:
        rospy.loginfo("Some Exception")
```