THE UNIVERSITY OF NOTTINGHAM MALAYSIA



An Intelligent Navigation System for Oil Palm Plantations

MOHAMMED AYOUB JUMAN, MEng

Thesis submitted to University of Nottingham for the degree of Doctor of Philosophy

ABSTRACT

In order to introduce automation to oil palm plantations as a solution to a reduction in workforce, a smart navigation system capable of operating autonomously was required. This thesis focuses on the creation of an intelligent navigation system for oil palm plantations. For the successful creation of the system, an object detection system to detect tree trunks, a path planner to calculate the optimum path to the tree and a trajectory controller to accurately move along the path was necessary. A robot rover was designed and built to test the created system, and novelties were presented in the object detection as well as the trajectory control systems to tackle the difficulties faced due to the plantation environment.

A novel tree trunk detection algorithm was proposed for object detection, focusing on oil palm trees in the plantation, as they were the key objects of interest. The algorithm used the Viola and Jones detector with a proposed pre-processing method along with depth information to tackle the issue of high false positive rates when the Viola and Jones detector was used on its own. The pre-processing method used colour space combination and segmentation to eliminate the ground not covered by trees and fed the resulting image into a cascade detector. The method had better performance when compared to both Neural Network and Support Vector Machine based detectors with a detection rate of 91.7% and had the lowest false acceptance rate overall, as well as a 97.8% detection rate during real time testing with the use of the robot, proving its high accuracy. Depth information obtained via the Microsoft KINECT sensor, resulted in a 100% detection rate of tree trunks within the sensor's range during low light conditions.

The nearest detected tree as well as the distance to it was obtained and marked as the goal point, while the surrounding obstacles were detected via ultrasonic sensors. This information was then passed onto the path planning system, created by a modified D*lite algorithm, to plan an optimal path to the tree trunk from the robot's current position.

The trajectory control system was based on the Enhanced Self Organizing Incremental Neural Network (ESOINN), which was able to produce exceptional trajectory control

without the use of a kinematic / dynamic model of the robot while consistently reducing errors during incremental learning. However, the ESOINN was only capable of discrete outputs, which limited the motions capable by the robot. A novel incremental unsupervised learning algorithm that is capable of producing continuous outputs, called IDW-ESOINN, was proposed by incorporating an interpolation method known as Inverse Distance Weighting (IDW) to extract outputs from three of the nearest nodes to the data point and combining them based on their distance to the data. The method also led to the creation of virtual nodes that enabled more nodes to be produced in empty areas during the next stage of incremental learning. The IDW-ESOINN had a higher performance accuracy than the original ESOINN as well as the Kalman filter method in simulated path following tests, with a consistently low RMSE, while having the lowest RMSE 60% of the time when compared with the other two methods. In real-time testing, the IDW-ESOINN outperformed the original ESOINN again, proving its functionality.

Based on the results obtained via the various tests done, it can be concluded that the created navigation system had a high level of performance, which could lead to fully autonomous robots being used in oil palm plantations one step closer to realisation.

LIST OF PUBLICATIONS

Published Journal paper:

1. M. A. Juman, Y. W. Wong, R. K. Rajkumar, and L. J. Goh, "A novel tree trunk detection method for oil-palm plantation navigation," *Comput. Electron. Agric.*, vol. 128, pp. 172–180, Oct. 2016

Published conference paper:

1. M. A. Juman, Y. W. Wong, R. K. Rajkumar, and C. Y. H'ng, "An integrated path planning system for a robot designed for oil palm plantations," in *TENCON 2017 - 2017 IEEE Region 10 Conference*, 2017, pp. 1048–1053

Papers under review:

1. M. A. Juman, Y. W. Wong, R. K. Rajkumar, K. W. Kow and Z. W. Yap, "An Incremental Unsupervised Learning Based Trajectory Controller for a 4 Wheeled Skid Steer Mobile Robot," in *Engineering Applications of Artificial Intelligence*.

2. M. A. Juman, Y. W. Wong, B. K. Puah and R. K. Rajkumar, "An Adaptive Learning Approach for the Self-Organising Incremental Neural Network," in *IEEE Transactions on Systems, Man and Cybernetics: Systems.*

ACKNOWLEDGEMENTS

This entire PhD would not have been possible without the love and support granted by my family, who gave me the opportunity and strength to pursue my dreams and to reach this stage in my life, and I am eternally grateful for that.

I would not have been able to complete this PhD, let alone have an inkling of how I could have achieved all the objectives, without the guidance and support by my supervisor Dr. Wong Yee Wan. I will always be thankful to her for guiding me on the correct paths of thought to accomplish the various aspects of this project, and for her patience in dealing with all the delays and failures that I encountered along the way. I could not have hoped for a better supervisor for this PhD.

I would also like to thank my co-supervisor Dr. Rajprasad Kumar Rajkumar, for his constant support in helping me through this PhD whenever difficulties were encountered.

This PhD could not have been completed without the help of numerous final year project students who assisted me in the collection of data, testing out methods and construction of the robot. My immense gratitude goes out to all of you: Ng Yuan Shen, Chiew Jia Hao, Chiu Yan Ping, Yeoh Zhi Hong, Yong Chung Li, Goh Lay Jian, Yap Zhen Wei, H'ng Cong Yuan and Puah Boon Keat.

Finally, I would not have been able to endure this PhD without the support offered to me by all my friends and colleagues I came across throughout this journey. For the moral support, motivations and help provided during these years, I am eternally thankful to all of them.

LIST OF CONTENTS

ABSTRACT i
LIST OF PUBLICATIONS iii
ACKNOWLEDGEMENTS iv
LIST OF CONTENTS v
LIST OF ABBREVIATIONS ix
CHAPTER 1: INTRODUCTION 1
1.1 Overview of Research 1
1.2 Problem Statement 2
1.3 Aim and Objectives 4
1.4 System Overview 6
1.5 Significance of Research 8
CHAPTER 2: LITERATURE REVIEW 11
2.1 Robotics in Plantations 11
2.2 Object Detection 16
2.2.1 Pre-processing and Feature Extraction 16
2.2.2 General Object Detection Techniques 19
2.2.3 Object Detection in Agriculture 23
2.2.4 The Viola and Jones Algorithm 27
(A) Haar-like Features: 27
(B) Integral Image: 27
(C) Adaboost Learning Algorithm: 28
(D) Training Method: 30
2.3 Path Planning 31
2.4 Trajectory Control Techniques 33

2.4.1 The Enhanced Self-Organising Incremental Learning Neural Network (ESOINN) Algorithm....40
2.4.2 The Inverse Distance Weighting (IDW) interpolation algorithm....44
2.5 Summary of Literature Review....45
CHAPTER 3: METHODOLOGY....47
3.1 Robot Rover....47
3.1.1 Chassis Design....47
3.1.2 Component Selection....49
(A) Sensors for Distance Measurement:....49
(B) Sensor for Movement Measurement:....49
(C) Sensor for Position Location:....50
(D) Camera for Vision System:....50
(E) Motors for Motion:....50
(F) Controller for Components:....50
(G) Main Processor....51
(H) Power Source:....51
3.1.3 Component Layout....52
3.1.4 Fully Constructed Robot Rover....53
3.2 Tree Trunk Detection System....54
3.2.1 Data Collection....56
3.2.2 Colour Image Processing....57
(A) Novel Pre-processing Method....57
(B) Cascaded Tree Trunk Detector....59
3.2.3 Depth Data Processing....61
3.2.4 Nearest Tree Localization....64
3.3 Path Planning System....68
3.3.1 Obstacle detection and mapping....68

3.3.2 Integrated path planning system 69
3.4 Trajectory control system 72
3.4.1 Training Data Collection.....73
3.4.2 ESOINN Initial Training.....76
3.4.3 ESOINN Online Incremental Learning 77
3.4.4 The Novel IDW-ESOINN Algorithm.....78
3.4.5 PWM output extraction.....84
(A) Distance Calculation.....84
(B) Radius Calculation.....84
(C) PWM Calculation 85
CHAPTER 4: RESULTS AND DISCUSSION.....87
4.1 Tree Trunk Detection System 87
4.1.1 OPTD System Training.....87
4.1.2 Detecting Tree Trunks on Static Images.....91
4.1.3 Real Time Detection during Linear Motion.....95
4.1.4 Tree Detection Using Depth Data.....97
4.1.5 Real Time Tree Detection during Varied Motions 99
4.1.6 Estimation of Distance to an Oil Palm Tree 100
4.2 Path Planning System 101
4.3 Trajectory Control System.....104
4.3.1 Testing the Capability of the ESOINN in Trajectory Control 104
(A) Obtaining the Optimal ESOINN Training Parameters.....104
(B) Testing of ESOINN Outputs in Simulating Paths 108
(C) Performance Comparison with other Methods.....110
4.3.2 Creation and Testing of IDW-ESOINN method.....113
(A) Approximation of Runge's Function.....113
(B) Distribution of Learnt Nodes after Initial Training 115

(C) Distribution of Learnt Nodes after Incremental Learning 117

4.3.3 Performance on Path Following.....120

(A) Performance Comparison on Simulated Path Following 120

(B) Performance Comparison on Real-Time Path Following.....124

CHAPTER 5: CONCLUSION 128

REFERENCES 130

LIST OF ABBREVIATIONS

AUV: Autonomous Underwater Vehicle

CNN: Convolutional Neural Network

DCT: Discrete Cosine Transform

DOF: Degree of Freedom

DWT: Discrete Wavelet Transform

ESOINN: Enhanced Self-Organising Incremental Neural Network

FFB: Fresh Fruit Bunch

FNN: Feed-forward Neural Network

HOG: Histogram of Oriented Gradients

HSV: Hue Saturation Value (colour-space)

IDW: Inverse Distance Weighting

IMU: Inertial Measurement Unit

LIDAR: Light Detection And Ranging (sensor)

OPTD: Oil Palm Tree Detector

PCA: Principle Component Analysis

PID: Proportional Integrative Derivative

PWM: Pulse Width Modulation

RBF: Radial Basis Function

RGB: Red Green Blue (colour-space)

RMSE: Root Mean Square Error

ROI: Regions of Interest

SIFT: Scale Invariant Feature Transform

SOINN: Self-Organising Incremental Neural Network

SOM: Self-Organising Map

SVM: Support Vector Machine

UAV: Unmanned Aerial Vehicle

CHAPTER 1: INTRODUCTION

The main aim of this project was to create an intelligent navigation system, capable of controlling a robot through an oil-palm plantation, so that it may carry out the tasks given to it. While the given tasks may differ, the key necessity was that the robot required knowledge on how to traverse through a plantation, which would be handled by the navigation system created by this project.

1.1 Overview of Research

Malaysia is the world's largest exporter and the second largest producer of palm-oil. With roughly 5 million hectares of land as oil-palm plantations and harvesting being highly labour intensive, the palm-oil industry has experienced a slight decline in recent years due to labour shortages as less migrant workers are coming in to fulfil the job vacancies in plantations [1]–[3]. It is estimated that 5-10% of fruit is lost each year as there is not enough workforce to harvest them on time. One solution to this problem is – automation.

By creating an autonomous system that is able to fill in for the shortage of man-power; fruit harvesting and palm-oil production can be increased. An automated system can work longer hours, even during the night, making it possible to cover more land and handle a larger workload than a single individual. Processes can be streamlined as well by coordinating timings between robots for the best efficiency. This would be cost-efficient as well, since apart from the initial cost and standard maintenance costs; no monthly wages need to be paid, making it beneficial for the planters in the long run.

A few areas were identified where an automated system / robot may come in use at an oil-palm plantation: applying fertiliser for the trees, applying pesticide and herbicide for insect and weed control and picking up of palm fruits from the ground. For the automation of these task, whatever machine used, would essentially require a navigation system as it would have to traverse the plantation.

1.2 Problem Statement

For the creation of the navigation system, focus was given to solving the issues that could affect the feasibility / performance of the navigation system, based on the working environment as well as financial constraints. An image of a typical plantation is shown in Fig. 1.1.



Fig. 1.1 - Plantation environment

After observing the working environment, three key areas were found that had issues that required resolving:

1. **The detection of tree trunks**: As the key object of interest was the oil palm trees, the tree trunks had to be detected before any of the other systems could work. As seen in Fig. 1.1, multiple tree trunks were present in each image, and there was no significant contrast between the trees and the ground, especially with the presence of parasitic plants growing on the trees, unlike those trees as seen in other similar works [4], [5]. At a certain distance, the trees also start to look as if merged together making individual tree detection difficult. Lighting in the plantation was also quite varied, with areas of low and high light levels due to the presence of an uneven canopy overhead. Detection at night would also be an issue as no light would be present.

2. **Trajectory control of the robot**: A robot assigned to the plantation would have to be controlled perfectly, so that any given path could be followed accurately. Since the plantation environment is ever changing, the robot would have to move differently depending on the given path [6]. This required the robot to be

able to learn adaptively / incrementally, as it was impossible to program in advance every single motion the robot could carry out.

3. **The design of the robot**: The robot design was crucial as the testing of the navigation system would depend on how well it could traverse the plantation. As seen in the images, there were no clear paths present between the trees, so the designed robot would require the ability to move in the muddy / grassy terrain, while having sensors on-board to be able to detect obstacles present, all of which should be cost-effective so that it was financially feasible.

1.3 Aim and Objectives

The aim of this project was to develop an intelligent navigation system, capable of operating in an oil palm plantation, by detecting the nearest tree trunk, planning an optimal path to get to the trunk, and travelling to it by using an incremental learning based trajectory control algorithm. To achieve this aim, the following objectives were set:

1. **Develop a tree trunk detection system**: The trees visible to the robot from its current position had to be detected, while also narrowing down on the nearest tree. The detection rate had to be above 85%, based on the average minimum detection rates from similar research [4], [5], [7], so that no trees would be missed by the system. The false acceptance rate should be kept as low as possible, as this value would determine how many non-trees are wrongly detected, which would lead to extra time wasted in eliminating these false detections. The system should be able to perform in different times of the day, even at night, and while the robot was moving at varied speeds.

2. **Develop a path planning system**: The system should be able to detect any obstacles in the surrounding that could prevent the robot from getting to the tree it was focused on. The planned path must be the shortest available path while new paths should be able to be recursively computed in case new obstacles were detected in the current path.

3. **Develop an intelligent trajectory control system**: For any given motion to be physically followed, the control of the robot's motors required an intelligent system capable of learning and carrying out new unknown motions accurately.

4. **Integrate all systems in to one and test its performance on a robot**: All the developed subsystems should be tested to check their performance, while all of them had to be integrated to create a whole successful working system. A robot rover had to be built that would enable all parts of the navigation system to be successfully implemented and tested in a plantation environment. The robot had

to be designed to suit the working environment perfectly while keeping to a low budget and containing all the necessary hardware and sensors to ensure its full functionality.

1.4 System Overview

To achieve the mentioned aims and objectives, the created navigation system was split into 3 key subsystems: the object detection, path planning and trajectory control systems, and a basic overview of the system created is displayed in Fig. 1.2 below.

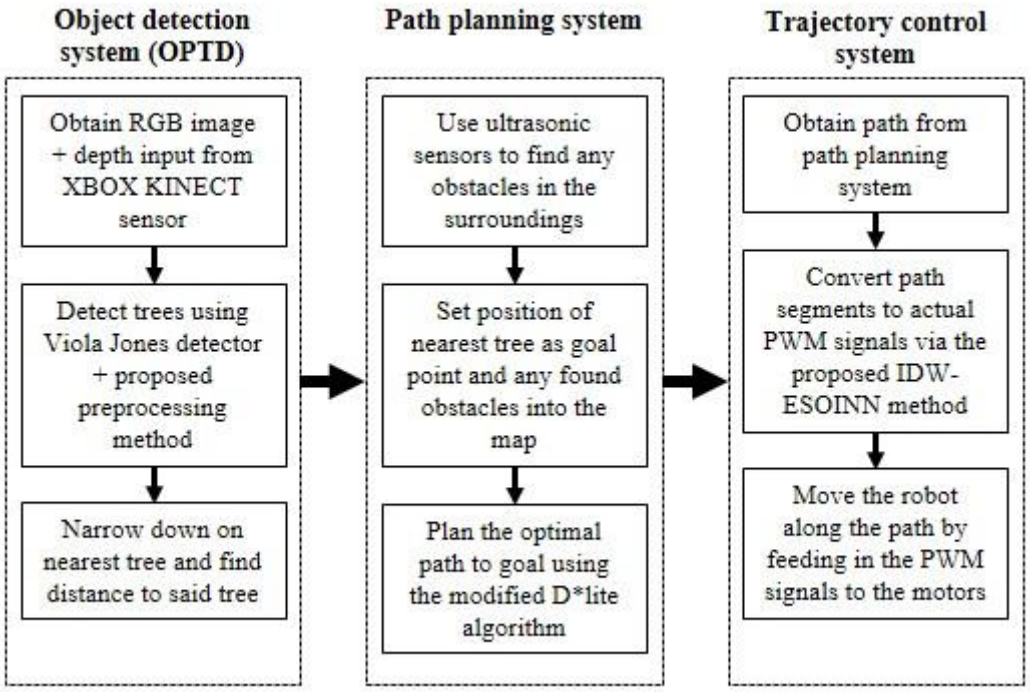


Fig. 1.2 - System overview flowchart

As seen in Fig. 1.2, each system worked independently, while passing the final output to the next subsystem when needed. As the key object in the plantation is the oil palm tree since all the tasks are performed around it, the object detection system focused on tree trunk detection. The input was obtained via the Microsoft XBOX KINECT sensor as both an RGB image and depth information. This was fed into a novel tree trunk detection method, created using the Viola and Jones cascade detector [8] with a novel pre-processing method created specifically for this project to eliminate the ground in the images to reduce false detections. The nearest detected tree was then narrowed down, while the distance to it was estimated via image processing, and then passed onto the path planning system. The entire tree detection algorithm was dubbed as the Oil Palm Tree Detector (OPTD) for the sake of convenience of reference. The path

planning system used this information to mark a goal point on a created map, while also marking any detected obstacles found via ultrasonic sensors mounted on the robot. The optimal path to the tree from the robot was then planned by using the 'moving target D*lite' algorithm [9] which was slightly modified to suit the robot, and this path was then passed onto the trajectory control system. The trajectory control system, split this full path into navigable segments, and converted those segments into the PWM signals necessary for motion by using a novel method (IDW-ESOINN), created by using the Enhanced Self-Organising Incremental Neural Network (ESOINN) [10] and an interpolation method called Inverse Distance Weighting (IDW) [11] to extract the outputs from the network. The extracted PWM signals were then sent to the robot's motors, to move the robot along the given path. These processes were repeated until the tree trunk was reached, while reprocessing was done at any stage if the robot had shifted beyond the given path or newer obstacles were encountered.

1.5 Significance of Research

The creation of an intelligent navigation system in this project, opens up options for the oil palm industry in automating their operations in their plantations. Recent developments in agricultural engineering in Malaysia for oil palm plantations have been focused on identifying the ripeness of oil palm fruit by using machine vision techniques [12], designing a 6 DOF parallel robot to climb a tree [13], and an oil palm FFB harvesting robot [14]. However, there is a lack of autonomous navigation development for palm oil plantations, which this project attempts to resolve.

During the creation of this navigation system, novelties were introduced in both tree trunk detection and trajectory control of the robot. Although others have done tree trunk detection before [7][15][4], there has been no significant research done in detecting oil palm trees, especially in such a low contrast environment. The created tree trunk detection system was based off the Viola and Jones [8] cascade detector, with the addition of a colour space based pre-processing method used to eliminate the ground from input images. The removal of the ground from the input images made a very significant difference in detections, as the lack of significant features on the tree trunks made the use of only a detector inaccurate, and also as the ground contained certain features that could be mistaken as features of a trunk, the use of the proposed pre-processing method helped resolve this issue. The created detection system performed extremely well on static colour images as compared to other methods such as SVM and Neural Networks, including the original Viola and Jones detector. In addition to this, the use of the pre-processing method, helped reduce the false acceptance rate, which would save the waste of processing time needed to eliminate them. Depth data was also obtained via the Microsoft XBOX KINECT sensor, which provided input on tree trunks during low light conditions, enabling detections at night and increasing detection accuracy at dusk and dawn when the light level was low. All the results proved that the proposed novel tree trunk detection system outperformed the original detector while maintaining a very high accuracy under varied lighting conditions as well as during motion.

For trajectory control, majority of the methods currently in use are based on using the kinematic or dynamic model of the robot to obtain the motion trajectory by calculating the relationships between the various forces acting on it [16]–[19]. Most of these systems focused on reducing the error of the predicted motion once the model of the robot had been obtained, which caused a heavy reliance on initially creating an accurate model of the robot. The trajectory control algorithm proposed during this project, was an unsupervised incremental learning system which enabled the adaptive learning of previously unknown robot motions, which essentially created a controller that became more accurate and capable as time passed. The advantage of this system was that the kinematic and dynamic model of the robot did not have to be calculated, as the algorithm was able to relate the necessary trajectory to follow to the movement to be done by the robot via the learning done by the network, by only requiring a set of initial motions done by the robot to start the learning process.

The proposed trajectory control system, dubbed as IDW-ESOINN, was created by adding the interpolation method known as IDW [11] to the ESOINN [10] algorithm, to create continuous outputs, which was previously unobtainable when only using the ESOINN method. The original ESOINN algorithm was only capable of discrete outputs, limiting the system to motions of only learnt nodes, which would make inter-node motions unable to be performed. The addition of IDW interpolation made use of 3 of the nearest nodes to the input data to interpolate the output data, thereby creating a continuous output, which also led to the creation of virtual nodes as this output data could be fed in as a new node during the next phase of incremental learning. The created IDW-ESOINN algorithm was used to obtain PWM outputs to control the robot, when an input was given in the version of the distance and radius the robot must travel by. Tests conducted on prior and post incremental learning, showed that the proposed method led to the creation of a better spread of nodes, especially in areas that were previously uninhabited, which directly increased the output resolution of the network. Tests also showed that the proposed method had exceptional performance as compared to the ESOINN and other methods. The system was also able to perform with extremely high accuracy as compared to other established methods, without any sort of kinematic / dynamic model of the robot, making the application of the system more flexible.

The following chapters of this thesis discuss the progress made in creating these proposed methods as well as achieving the mentioned objectives and the creation of the intelligent navigation system. Chapter 2 makes note of the related work done by researchers over the years, making note of methods used in situations similar to that of this project. Chapter 3 explains the methods by which the project objectives were reached along with the algorithms created and used, while Chapter 4 shows the results obtained when testing the created subsystems and these results are then compared and discussed with relation to other similar methods as well. Chapter 5 concludes the thesis, by summarising the key points of the overall created system.

CHAPTER 2: LITERATURE REVIEW

This chapter comprises of 4 sections, to make clear distinctions of the past and current work related to the methods used in this project. The first section notes the development of robotics and automation in plantations over the years. This is followed by a section dedicated to the favoured object detection algorithms currently being used. The third section focuses on the path planning aspect of this project, while the fourth section is dedicated to trajectory control and the various learning methods used to achieve it. The final section comprises of a summary of the key points extracted through the literature review, which impacted the decisions in this project.

2.1 Robotics in Plantations

Provision of food is one of the top priorities to mankind, resulting in the survival of the agricultural industry even with the development of countries with higher focus on the manufacturing and service sectors. It has always been a labour intensive field, with tasks such as planting / sowing seeds, maintaining health of the crop via irrigation and fertiliser, protecting the crops by using herbicides / pesticides and finally harvesting the crops when the time is right. Over the years the industry has greatly benefited with the use of machines such as tractors and large scale harvesters, yet there is still a lack of autonomy when looking at a majority of the on-field operations. Due to the monotony of most of the work and the decrease in the number of people willing to work in this sector worldwide, autonomous systems and machines provide a very real, efficient and hopefully cost-effective alternative. The reasons behind the lack of autonomy are due to the unpredictability in the field as there are uncontrollable variables, but a solution to this problem is getting nearer with the increase in research dedicated to this field.

A harvester for trees, intended especially for cherry harvesting was manufactured by Tanigaki et al. [20], meant as a solution to a short harvesting season and shortage of labour, the robot used rails as a guideline for the platform and an end-effector to do the fruit picking. The trees were planted in rows to facilitate robotic assistance and the fruit was picked by using suction to grasp the fruit while the stem was cut. De-An et al. [21] created an apple harvesting robot, both tested indoors as well as on-field. This was a

crawler type vehicle, with the end-effector having two spoon-like grippers to suit the shape of the apples for better surface contact. Results from laboratory tests showed an 86% picking rate and outdoor tests had a 77% pick rate. In both the harvesting robots, issues were encountered where fruits that were blocked by branches could not be detected and/or picked. Silwal et al [22] also created an automated harvester for apples, with a 7 DOF manipulator, which had a success rate of 84% in picking fruit in an orchard, with an average picking time of 6.0s per fruit. Martinez et al [23] created a system for automated control of an olive harvesting tree shaker, to help optimize the harvesting process. The created system had an accuracy of 92.9%, and a reduction of 27.3% in the time taken to grab a trunk, leading to a higher efficiency. Though the harvester had to be driven close to the tree, the trunk could be detected, grabbed and shaken autonomously with their system.

Bakker et al. [24] designed a robot for use in sugar beet fields with the intention of creating an autonomous navigation system for the robot platform. The robot was four-wheel steered and four wheel-driven, making control more complex yet giving options on various trajectories. Space to mount implements was available if needed, although in the referred design only one hoe holder was attached. The work done by Xue et al. [25] and Hiremath et al. [26] were focused on row following algorithms in corn fields and maize fields respectively. Xue et al. conducted tests on the use of variable fields of view, where the view could be changed by adjusting the pitch and yaw of the attached camera. This set-up was backed by the idea that changing the view would be useful in situations where gaps were present in a row or when the robot reached the end of a row where no plants were visible ahead. Changing the view would give the control system options to move past these blocks. Hiremath et al. focused their work on a sensor model for a 2D range finder (LIDAR) to overcome problems encountered by vision systems due to illumination changes in outdoor environments. The use of the sensor proved useful in developing a robust navigation system to be used in a semi-structured agricultural environment.

Autonomous navigation has not only been targeted for small scale vehicles, but for bigger ones such as tractors as well. Kayacan et al. [27] worked on the trajectory control of an autonomous tractor, while Yin and Noguchi [28] developed a target following system for a field robot, applied to an electric tractor type vehicle. Their work proved

that precise control of even larger agricultural vehicles was possible, widening the choices available for automation in the industry. Vidoni et al. [29] noted in their paper comparisons between different wheel configurations of robots operating on side-slopes. This research is invaluable as certain agricultural environments are situated in variable terrains. Through their work it was assessed that 3-wheeled vehicles were the most manoeuvrable yet also the least stable and the vehicles with highest stability were those with an articulated chassis. Ortiz and Olivares [30] designed a robot in the shape of a monster truck named Yeti for vision based navigation in plantations. The robot had 4 wheels, with the front and back wheels constructed to be able to be rotated at will, so as to maintain a constant centre when the robot was turning.

Cheein and Carelli [31] provided great insights into the current unmanned robotic service units in use in the agricultural industry. Along with reviews on techniques used and a case study on a working robot in olive groves, they mention that key abilities of automated robots in agriculture fall under four categories: guidance, detection, action and mapping. They stressed that to have a successful system; all four parts are required and require connectivity for smooth operations. The review done by Bechar et al [6] recognised the different types of robots present under current use as well as mentioning the difficulties faced in applying robots to outdoor unstructured agricultural tasks. They mention that smaller robots are preferable for applications as they use less energy, are more manoeuvrable and have a smaller impact on soil conditions, while being considerably cheaper than large vehicles. They also recognised the necessity for better subsystems as well as their integration for future commercialization of agricultural automation. Hajjaj and Sahari [32] provided another review on agricultural robots, going through their structure and significance to their respective applications.

It can be seen from all the literature that has been mentioned that a common trait for most of the constructed robots is the use of a 4-wheel design, either with a steering-column system or a skid-steer based system, with the robot size varying depending on the application. The 4-wheel design has more robust characteristics while the skid-steer system enables more motion opportunities due its higher manoeuvrability in smaller spaces, as the robot could be rotated on the spot if needed, which was quite a useful characteristic for the robot needed for this project. Furthermore, the robot carried all the

required equipment, sensors and processors on-board so that it could function as an isolated unit.

Bringing the focus to oil palm plantations, current work mainly focuses on the identification of the ripeness / health of FFBs such as the automatic grading machine for FFB by Makky and Soni [33] which was able to split FFBs into an accept / reject pile based on a K-means clustering image processing system. Kassim et al [34] created a K-means clustering based growth modelling technique to identify the maturity level of the fruit in the field as compared to in the mill where it was too late. Makky [35] developed a portable inspection system that contained an inspection chamber that an FFB could be inserted into, which then used machine vision to determine the ripeness of the fruit. Pamornnak et al [36] created a conveyor system with a KINECT sensor to grade FFB to determine acceptable fruits, with an accuracy of 83%, eliminating the need for an external inspector. Though there are several research articles to be found on the assessment of the oil palm fruits and more work on automating the extraction processes in mills, there is a lack of research being done on the introduction of automated robots to the oil palm plantations. Current commercial products are more focused on creating devices to better harvest and harvest the fruit than a navigation system for operating in the plantations.

A survey was also done to assess the available range sensor options currently available according to their specifications and cost. Range sensors for measuring distances were primarily of 3 types: ultrasonic sensors, laser array sensors (LIDAR) and infrared sensors. The pros and cons of each type are mentioned in Table 2.1 below:

Table 2.1 - Pros and Cons of various range sensors

Sensor type	Pros	Cons
Ultrasonic	Not influenced by light conditions Good range Low price	Inaccurate at close quarters Accuracy decreases with range Absorbed by certain materials
Laser array (LIDAR)	High accuracy 3D sensing Wide scanning area	Influenced by lighting High price
Infrared	Accurate at short ranges Low price Low power consumption	Influenced by lighting Low range

Considering the available budget, ultrasonic sensors were decided as the best choice, as the plantation environment would generally have a high amount of light during the day, eliminating pure light based sensors. Although there were some models of LIDARs that were powerful enough to have good detection capabilities even during regular daytime lighting, the price was considered to be too high for the application of this project.

2.2 Object Detection

As the solution to the problem: what is the machine seeing, object detection has become a crucial part of autonomous systems and has been widely researched for a number of years. It is the process of teaching a system what an object looks like, then requesting the system to identify the learnt object in new situations. Object detection systems have a few key stages that are usually set for a majority of techniques: pre-processing and/or feature extraction, the detection algorithm and post-processing; with the detection algorithm being the most important stage. The chosen algorithm usually is the core component in determining the speed, robustness, complexity and processing power required of the system.

2.2.1 Pre-processing and Feature Extraction

In most cases, images and data that are fed-in to object detectors tend to have certain problems in their original state. The most common problems are caused by: presence of noise, bad illumination, low contrast and high dimensionality of data. The first three are generally handled by pre-processing the data while the latter is handled by feature extraction performed on the data to reduce the dimensions. Pre-processing steps are generally centred on making the object of interest stand out as compared to the background or in certain cases to make the image texture even, so that proper detection can be done. Majority of the methods are quite simplistic in nature as they are light computations done to reduce the necessity of heavy computations later on.

A colour image is usually in a RGB format where the image is made of layers of pixels corresponding to how red, green or blue they are. Colour segmentation is a simple step performed where colour information of the object / background is targeted. This can be the removal of certain colours, conversion and / or combination of different colour spaces or thresholding so that only values above or below a certain limit are passed on. Vidal and Amigo [37] have described in detail various segmentation methods that are applied for situations. A well-used method for increasing image contrast over the years has been histogram modification and histogram equalization. The main idea behind both the methods was to reassign the intensity values of the pixels to suit the needs of

the user. Global adjustments such as distributing the intensity values across the range of values goes under histogram equalization, while local changes performed on certain ranges is histogram modification. Hongbo and Xia [38] used histogram modification techniques to conditionally enhance the contrast on images across numerous samples.

Illumination pre-processing is another key area, mainly in situations dealing with environments where the lighting is not controlled, especially outdoors, where some regions or the entire image gets lighter / darker. Simple situations can be resolved by adjusting the overall / partial pixel intensities for an image but in more complex situations where shadows and light are intermingled over an object, algorithms dedicated for illumination enhancement are required. Han et al. [39] have done a comparative study on illumination algorithms for pre-processing in face recognition, with tests conducted on 12 different methods over 4 different face image databases.

Once pre-processing is done, the process moves onto feature extraction. A point to note is that not all systems use pre-processing as well as feature extraction. Some use either one of the methods and in certain none at all, although this is rare as the data and the system have to be almost perfectly compatible. Feature extraction methods have one base similarity. They all perform computations on an image to extract features from it, letting the user extract significant features so that the image can be presented in a new manner with reduced data, thereby reducing dimensionality. Common feature extractors that have been used over the years are in general statistical tools or mathematical models that act as filters. Two mathematical operators that have been adopted as feature extractors are the Discrete Wavelet Transform (DWT) and the Discrete Cosine Transform (DCT), while the statistical method widely used is known as Principle Component Analysis (PCA).

Generally on an image, a 2D DWT applies a combination of low pass (LP) and high pass (HP) filters to split the image into four parts, each becoming a quarter of the size of the original image. The four parts are referred to as LL, LH, HL and HH where each part is a combination of operations done on rows by the first filter and then columns by the second. Noisy components in general have high frequencies so the key data is contained in the LL part. The work done by Bhanari et al. [40] as well as Huang et al. [41] made use of the DWT for contrast and brightness enhancement and feature

extraction respectively. A DCT works by expressing sequences of data points as a sum of cosine functions. Once a DCT is applied to the image and the coefficients extracted, either this entire feature set is fed in, or in other cases coefficients in a certain range are extracted as significant features. The method has been used for differing situations such as human detection [42], copy-move forgery detection [43] as well as satellite image enhancement [44].

Finally moving onto PCA, the method performs an orthogonal transformation on possibly correlated data, to a set of linearly uncorrelated values deemed as principle components. Most methods choose to extract those values with the highest variance by choosing all the values that account for the decided amount of variability of the data. These components are the eigenvectors obtained through PCA and they produce the features from data by multiplying the data values with the vectors. Kuncheva and Faithfull [45] used PCA as a feature extractor for detecting change in unlabelled data, although they used the principle components having the least variance as these values were more sensitive to change. A different paper published by Oh, Yoo and Pedrycz [46] showed the use of PCA used in tandem with another statistical method known as linear discriminant analysis (LDA), as pre-processing for a neural network based face detector.

A highly relevant paper would be the survey done by Gao et al. [47] that makes note of various object detection techniques, pre-processing methods and feature extraction algorithms. Although the paper was specifically for planetary terrain classification, clear explanations and examples were given to the various methods mentioned above, along with other feature extraction methods such as SIFT, HOG and Haar-like features.

2.2.2 General Object Detection Techniques

Over the years multiple different techniques have emerged for object detection, each approaching it in a different way. Due to the similarity between object detection and pattern recognition, many techniques have been used interchangeably between the two categories, lead examples being Neural Networks and Support Vector Machines. Neural networks have been used widely for various problems since its start over 50 years ago with the introduction of the Perceptron rule and has made great strides in its changes over the years [48]. The core concept was to mimic the process of the human brain, where single neurons work together with multiple others to perform complex tasks. A single perceptron sums together the values of all the inputs multiplied by their corresponding weights, while a multi-layer perceptron is a combination of multiple perceptrons in more than one layer, forming a neural network.

Training is done by feeding in training data and adapting the weights according to the error between the network output and the expected output. A key algorithm for the weight update procedure was the introduction of the backpropagation algorithm [49], that is still used to this day. Starting off at pattern recognition, neural networks have moved onto other categories like machine learning, data fitting and prediction. Recent years have seen networks applied to various situations from face, vehicle and general object detection.

Kouskouridas et al. [50] made use of neural networks trained with a Radial Basis Function (RBF) [51] for the task of 3D object pose estimation. The system was designed to judge the orientation of objects in images as well as track new unregistered objects that appeared for the first time. Features were extracted from the data by using a fuzzy clustering technique then fed into the network for training and testing. The choice of using the RBF neural network was just, as the input-output mapping was found to be non-linear which is a scenario that RBF networks excel at. Huang and Do [52] also used a similar type of RBF neural network for the purpose of motion detection in dynamic scenes, for the same reason of non-linear input-output mapping.

A specific type of neural network known as the Convolutional Neural Network (CNN) is especially used on images for object detection [53]. A CNN has multiple layers, and the training process identifies image features in every layer, which is then passed onto the next, enabling a smaller scale of processing for further layers. Zhou et al. [54] used a CNN to identify trees based on their leaves and were able to differentiate between 25 tree species with an accuracy of 87.57%. Aparna et al. [55] created an automatic tree counting system for coconut trees by using images from a UAV to train a CNN. The network had an accuracy of 90%, though it could not differentiate between overlapped trees. Fu et al. [56] used a variant called Faster R-CNN to detect kiwi fruits in the field. The trained network had an overall recognition rate of 92.3% and was even able to separately identify fruits from clusters. Branson et al. [57] focused on the creation of an up-to-date catalogue of trees in a city by using aerial and street view images from Google Maps. The network they created was able to detect over 70% of street trees while assigning them to 40 different species and was also able to keep track of the changes to the trees over time. Though CNNs do have a high degree of accuracy when classifying images, they do have a disadvantage of requiring a high number of training samples, ranging from a few thousand to millions, while also needing high processing power based on the input size during training.

Support Vector Machines (SVM) was introduced by Vapnik in 1979 in Russia and the work was translated to English only in 1995 by Cortes [58]. Having its roots in statistical theory, SVM non-linearly maps input vectors to a high dimension feature space. A linear decision surface with special properties ensures the high generalization ability of the network [58]. Pang et al. [59] made use of a linear SVM with a modified approach of the sliding window technique that was composed of a window that was moved over parts of the image exhaustively, producing sub-windows to be fed into the detection stage. The proposed solution was named distributed object detection (DOD) with the main aim of reducing computational costs by decreasing the size of features in the sub-windows. Results proved a high efficiency for the method designed during applications for hand, face and pedestrian detection, each in separate cases. Kurtulmus and Kavdir [60] created a system involving SVM for detecting corn tassels for agricultural applications and the algorithm resulted in detection rates of 81.6% during testing.

A method that has received more attraction within the last 10 years is saliency based methods. The method revolves around locating the most salient object in an image, which is basically the object at the foreground that is most focused upon. Algorithms in this category attempt to box-in the most salient object of the image, while keeping as much of the background out of the selected area. A salient object is detected by first obtaining a saliency map, which is done by performing computations on multiple different features obtained from the original image at different scales. The features generally range from colour, intensity to motion and texture. Once a saliency map is obtained, detection is performed on it to determine which one is the object or what area the object is within. In his paper, A. Borji [61] explained in detail about saliency, as well as introducing a new dataset and a baseline model for salient object detection. The new model was simple and computationally efficient than other similar models with very good object detection characteristics. In most cases the object was clearly defined in the final output with very less noise and background. Shi et al. [62] developed a saliency based method with the intention of obtaining better results from saliency maps with cluttered information. Their method proved to be highly efficient by obtaining multiple iterations of a saliency map through modification of the previous iteration. A more accurate window for surrounding the salient object was obtained compared to other methods and the speed was twice as much as the second fastest method.

Another common method used for object detection is by making use of a cascade type architecture. Generally a cascade type architecture has multiple stages, with each stage being composed of various features, where an input is considered positive / acceptable if it passes through all the stages. If an input does not meet the necessary thresholds set at any stage through its features, it is rejected and does not move onto a forward stage. The stages become more complex as they progress, thus systems were more efficient using this structure as negative inputs could be rejected early on. Cascades were generally combined with boosting type learning algorithms that led to constructing strong classifiers by combining multiple weak classifiers. Simply said, combining ten classifiers with a 40% error rate and a 99% detection rate would give a final classifier with a 0.0001% error and 90% detection rate, as the performance of each classifier is multiplied by the performance of the next one.

One of the most popular cascade type detector was developed by Viola and Jones [8], by creating a new learning algorithm based on the Adaboost learning algorithm [63] to select a small number of key features from a large set, as well as creating a new set of features called Haar-like features based on the Haar wavelet function. They also developed a new image representation called the integral image that made the computation of features extremely easy. The final algorithm was mainly used as a face detector which is still used to this day, with a detection rate close to 93%. The algorithm has been modified for various uses and improved over the years by addition of new filters as features, mainly by Lienhardt and Maydt [64] in 2002 by adding rotation to the original filters and recently by Vural et al. [65] as well as Park and Hwang [66] to give improved results. The Viola and Jones algorithm succeeded mainly due to its fast computation in detections and reduced complexity, making it a first choice for applications where speed was a necessity.

Other recent work in cascade detectors was by Kim, Jung and Chung [67] in applying face detection to facial expression recognition. Their method used the Viola and Jones method, but with their own features instead of Haar-like features and results indicated accurate detections with a high detection speed compared to other methods. The system was also able to recognise a variety of emotions, from happiness, sadness, disgust, anger to a total of 7 emotions. In 2014, Hoang, Le and Jo [68] developed a boosted classifier with the use of HOG filters and a SVM for pedestrian detection. Their attempt was to combine the efficient learning model of SVM and the early rejection of negatives of the cascade architecture. The resulting system had higher accuracies than a pure SVM approach and had double the speed as well. Other detection methods used in research include fuzzy-based classifiers [69], Markov models [70], Bayesian based methods [71] as well as decision trees [72].

2.2.3 Object Detection in Agriculture

With the development of sensors and machine vision techniques, several agricultural vehicles have been upgraded for automated navigation, harvesting and mapping of various sites. Obtaining readings of the working region is a key criterion for autonomous functionality, as this opens the way for object detection and mapping, both crucial processes to provide references for navigation. Cameras provide a low cost solution for obtaining information of the immediate environment by means of intensity and colour data, while range sensors such as sonar and lasers are used to obtain both 2 and 3 dimensional depth data.

The semi structured nature of orchards and plantations make autonomous navigation possible with the use of basic machine vision techniques and sensors. Using a planar laser scanner on a mobile robot, row by row navigation can be carried out. In these cases, the scanner is able to obtain range information of the trees on both sides of the row, enabling the mobile robot to centre itself and navigate to the end of the given row. Bayar et al [73] used a planar range finder to align their robot to orchard rows, focusing more on the row like arrangement than individual trees. Gimenez et al [74] used similar methods, although their system required prior knowledge of the absolute points of the end trees of the rows.

Although obtaining range information from planar sensors is useful for knowing the distances to obstacles and trees, they do not provide sufficient data for creating a good reconstruction of a given area by themselves. In certain cases, planar data is not sufficient to detect obstacles, leading to the need of more range data to make accurate assessments. As a result, the use of 2D LIDAR sensors to obtain 3D point clouds has increased in popularity in recent years. Given the high number of data points for each square meter, complete scene reconstruction is entirely possible, leading to higher chances of accurate detections. Sanz et al [75] used a 2D terrestrial LIDAR scanner for characterising tree crops by scanning each row on both sides via a tractor mounted sensor to obtain two 3D point clouds. The obtained data enabled them to calculate accurate leaf area density of the present trees. Similar work was done by Rosell et al [76] to obtain large amounts of plant information, giving a high correlation coefficient

of 0.976 between trees reconstructed using data points and manual measurements. Forsman and Halme [77] displayed the use of dense range images to reconstruct trees by representing them as cylindrical columns in scans of large scale environments. Their reasoning being that accurate mapping would enable virtual testing of forest working machines. Zhang et al [78] used a 2D LIDAR to generate maps of orchards to simplify autonomous navigation. Their system required marked posts as artificial landmarks for better judgement of row ends, producing results with only 19-26cm mean distance error.

Other research has used machine vision techniques for identification in agricultural environments, even without the use of sensors. Auat Cheein at al [79] tackled precision agriculture mapping with the use of SVMs for detecting olive stems from images obtained via a monocular vision system. Once the trees were detected by the system, a laser range sensor was used to give the distance of the stem to the robot. The absence of weeds and a clear contrast between the ground and the trunks made the use of a pure vision based system suitable for this scenario. Lu and Rasmussen [7] used an omnidirectional camera and contrast templates to detect tree trunks by relying on the attribute where trunks appear as straight lines with a high contrast at both sides. Shao et al [15] used colour features of the trunk from the L*a*b colour space as well as shape features by means of the Hough transform to detect trees. [4] furthered their work with the addition of a back propagation neural network to segment the colour marks from the L*a*b colour space before performing the Hough Transform. Torres-Sospedra and Nebot [80] used artificial neural networks for the detection of weeds in orange groves enabling their system to detect weeds with a very high performance rating.

Though there are several systems that make use of sensors or cameras by themselves, certain environments / situations require more data than either can provide individually. The solution for these cases is to make a system that combines the information obtained from both sensors and cameras so that the lack of information from one input is made up by the other. Jaakkola et al [5] worked with a multi sensorial system for mobile tree mapping, though their method was unable to distinguish between trees and other pole type objects. Zhong et al [81] used sensors along with a CCD panoramic camera to incorporate texture information into the reconstruction of the 3D model of the environment. Shalal et al [82] were able to detect trunks with 96.6% accuracy with the

use of a camera and laser scanner, similar to the previous systems, highlighting the advantage of using multiple input devices for increased detection accuracy. They predicted that although their system had high accuracy in the orchard, it might have poor results if the trunks were occluded with leaves / fauna. They furthered their work by using the tree detector for mapping and localisation by using the trunks as key points in the map [83]. Chen et al [84] used multiple cameras and ultrasonic sensors to detect orange tree trunks in an orchard with a HOG-SVM based classifier, with a detection accuracy of 95.49%. The camera and sensor combination also enabled them to localize the robot based on the distance to the trees, reducing the error of self-localization to an average of 62mm.

With the introduction of the Microsoft XBOX KINECT sensor that incorporates both vision and depth inputs, the possibilities of its usage for research purposes have increased over the recent years. Though mainly used for indoor applications due to the interference of natural light with the infrared light used by the depth sensor, there are a few cases where it has been used in outdoor / agricultural settings. Nissimov et al [85] used the KINECT in a greenhouse environment for obstacle detection via the use of depth and colour information. The slope of the environment ahead was calculated using depth data and if the slope was too large to be classified as ground, colour and texture features were used to classify the obstacle into various categories. Though not in an agricultural setting, Hernandez et al [86] used the KINECT for obstacle detection outdoors for autonomous vehicle navigation. They managed to circumvent the problem of natural light by placing the KINECT close to the ground and tilting it to face at a downward angle, reducing the chances of interference and focusing on the obstacles at close quarters. Majeed et al [87] used the KINECT V2 along with a CNN called SegNet to detect and segment trunks and branches of apple trees with an accuracy of 92 and 93% respectively.

Observing the mentioned research in object detection, it can be concluded that the created system can have inputs of either pure colour or in the form of depth images, while processing is done prior to and / or post detection by the main algorithm. For the detection system in this project the entire focus was given to the detection of oil palm tree trunks, as this was the key object of interest in the plantation and the focus of whatever tasks have to be performed at a plantation. To achieve the detections, a novel

pre-processing method was created which focuses on reducing the image size by eliminating the ground to the area that is assumed to have trees, which is then fed into the detector for further processing. The method works by using colour space combination to enhance the contrast between the ground and other objects and then cropping the image to remove ground areas. Varying lighting conditions and the low contrast between tree trunks and the background in visual information made the use of other sensory information a necessity for improving the accuracy of the system.

In keeping to low costs, the KINECT sensor was used for obtaining video input as well as depth information, instead of having two separate devices as inputs. Colour images could be used when the light was too strong for depth input, and depth information could be used when there was no light, while a mixture of the two could be used in low light conditions. The detector used on the video input was created by Viola and Jones [8], chosen for its cascade nature, which eliminated unlikely inputs in the early stages of the cascade. It was computationally lighter compared to SVM and neural networks, and was a more suitable choice than using saliency based methods. In almost all views in the plantation, there were numerous amounts of trees which appeared in the background, which made saliency based methods less reliable than situations where the most salient object could be easily seen. The cascade structure also meant that negative images could be discarded without needing computation by the entire algorithm, saving processing power and giving faster detection rates. Furthermore, the depth data from the KINECT was used in tandem with the detector to enable more accurate detections. The fully created tree trunk detection was named as the Oil Palm Tree Detector (OPTD). For reference, an outline of the Viola and Jones algorithm is given in the subsection that follows so as to better understand the created detection system.

2.2.4 The Viola and Jones Algorithm

Understanding the Viola and Jones algorithm requires knowledge of three different parts that combine to form the technique: Haar-like features, the integral image and the Adaboost learning algorithm.

(A) Haar-like Features:

The original Haar-like features proposed by Viola and Jones are given in Fig. 2.1.



Fig. 2.1 - Original Haar-like features

The features were computed by the taking the difference of the sum of the pixels in the different regions. Each feature had unique situations where they were more effective. The two left-most features worked best at detecting edge features while the three square feature worked well with features where the centre was a different colour, for example the nose and cheeks region of a face.

(B) Integral Image:

An integral image was a new representation of an input image with the same dimensions as the original. Each pixel in the integral image represented the sum of the pixels situated above and to the left of the current position, inclusive of the current pixel.

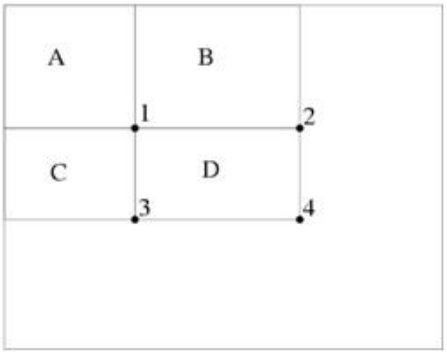


Fig. 2.2 - Computing the sum of pixels using the integral image

Using Fig. 2.2 as a reference, obtaining the sum of pixels in D would be simply computing D – (B + C – A) which is equal to 4 – (3+2-1), showing that one rectangle area can be obtained by just 4 references. Similarly for 2 rectangle features 6 references are need, 8 for three-rectangle and 9 for four. This simplified the computing of features as the integral image needed to be computed just at the beginning, making multiple feature computations relatively fast.

(C) Adaboost Learning Algorithm:

All the features for a given image are obtained by applying each feature at each pixel position, changing locations till all the areas are covered, then changing the feature size and repeating the process until the largest possible feature is obtained. For a 24x24 pixel area, the full set of features number around 180,000. Applying this many features for detection tasks is unrealistic, which is where Adaboost comes in. The Adaboost algorithm for classifier learning is given below:

1. Given example images (x_1,y_1), … , (x_n, y_n) where y_i = 0,1 for negative and positive examples respectively.
2. Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for y_i = 0, 1 respectively, where *m* and *l* are the number of negatives and positives respectively.
3. For t = 1, … , T: Normalize the weights,

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}} \quad (1)$$

   So that w_t is a probability distribution.

4. For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t,

$$\epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right| \quad (2)$$

5. Choose the classifier, h_t, with the lowest error ϵ_t.
6. Update the weights:

$$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i} \quad (3)$$

Where e_i = 0 if example x_i is classified correctly, e_i = 1 otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.

7. The final strong classifier is:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & otherwise \end{cases} \quad (4)$$

$$where\ \alpha_t = log \frac{1}{\beta_t}$$

In simple terms, at every iteration, Adaboost applies all the features to the inputs and chooses the feature that gave the best result, adding it to the main classifier. The final classifier is then run on the input and the weights updated. The process is repeated again, adding features until the final classifier has reached the false positive rate set by the user.

Viola and Jones modified this boosting algorithm to incorporate a cascade structure. A cascaded detector has multiple stages that form a degenerate decision tree. Only a positive result from the previous stage will lead to the computation of the sub-window at the next stage and a negative result at any stage means immediate rejection of the sub-window. While the detector has an overall detection rate and false positive rate, each stage has them as well. The modified algorithm is explained via the flowchart in Fig. 2.3 below:

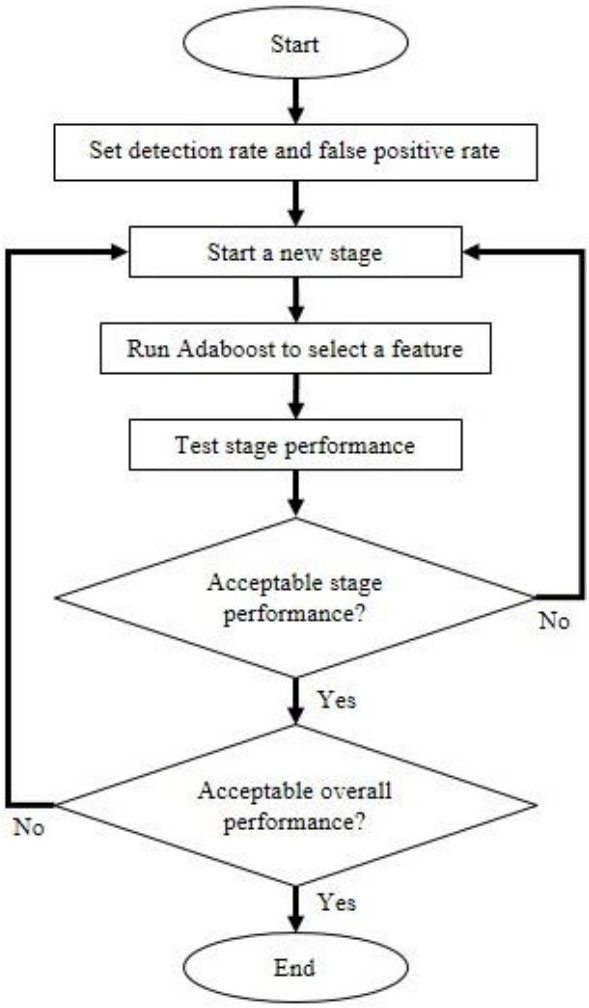


Fig. 2.3 - Flowchart of the Viola and Jones algorithm

(D) Training Method:

A Viola and Jones detector is constructed by feeding in positive and negative images as well as setting the necessary parameters by the user: detection rate, acceptable false positive rate and number of stages. To clarify, detection rate is the rate at which the object in question must be correctly detected at, the false positive rate is the number of non-objects that have been mistakenly labelled as objects, and the number of stages sets the number of cascades that the detector has. The choices of parameters influence the end performance of the detector as they are interconnected. As the number of stages increase, the false positive rate drops, yet this also results in a drop in the detection rate and increased computation time as there are more features to be computed. Yet small number of stages may not give acceptable detection rates. A point to note is the number of features per stage increase with the stage number as more poor features are needed to obtain the same performance as the better features are used by previous stages.

2.3 Path Planning

To introduce autonomy to an environment where the machine is required to move from point to point, it requires a navigation system capable of controlling this motion. For autonomous navigation to function, a path planning method that computes the ideal path to travel is a necessity, as the system needs to know the best way to reach a given end point either in the shortest available time, or with the most stable route.

One of the simplest path planning algorithms developed was the Dijkstra algorithm [88]. It found the shortest path to the goal through iterations of expanding neighbour nodes in a graph. This method was guaranteed to find the shortest path to the goal, though it expanded unnecessary nodes along the process, which increased the computation time. Hart et al. [89] introduced an approach to improve the formal mathematical theory of graph searching by using heuristic information from the objective of the problem. The path finding problems were approached and solved by using both a mathematical and heuristic approach. It combined these two approaches together and presented the A* searching algorithm that could use heuristic information to find a minimum cost path through the graph. This approach significantly boosted the computation time of path planning.

However, algorithms such as A* and Dijkstra completely re-planned the path if there were changes on the initially planned path and would repeat this process if there were more changes. This repetitive process accumulates computational time resulting in inefficient performance in large domains with rapid changes. Stentz [90] presented the focused D* algorithm to solve this problem by repairing the path when the initial planned path was blocked. It involved manipulating the status of nodes and used previous search information to repair the path. Sven et al. [91] also used a similar method to re-plan, though it was algorithmically different. It presented an incremental version of the ‘A* search’ algorithm, called the ‘lifelong planning A* search’. It also used previous search information and manipulated the status of nodes to repair the path but with a more simplified code structure.

This code structure was later used in implementing D* lite [92], a fast re-planning algorithm for navigation in unknown environments. It comprised the same searching method as A* search, but unlike A*, it searched from the goal point to the starting point.

For path planning to function accurately, accurate obstacle detection techniques are essential in planning a collision-free path efficiently. With the development of machine vision techniques and various sensors, accurate mapping of the environment was made possible. The KINECT sensor was used to obtain the depth data and colour information of the surrounding environment for obstacle detection in a greenhouse [85] and in an outdoor environment for obstacle detection for an autonomous vehicle [86]. Borenstein and Koren [93] installed two ultrasonic sensors set to swivel, in front of a nursery robot to map obstacles and perform obstacle avoidance. Kang et al [94] used both a wide vision camera and infrared sensors for a robotic vacuum cleaner to detect obstacles. As mentioned in the previous section, others researchers [5], [81] used more detailed objection detection techniques to detect and identify goal points or obstacles to provide highly detailed information about the surrounding environment.

The D* lite algorithm variant created by Sun et al [9] called ‘moving target D*lite’ was chosen as the path planning algorithm for this project due to its faster computation time compared to other similar algorithms that were mentioned in this section. It exhibited the same behaviour as ‘A* search’ in expanding to nearby neighbours, though it started its search at the goal point instead of the starting point and included an additional key condition in choosing the priority node as a successor to expand. D* lite had advantages over A* search in terms of efficiency in vertex expansion and computational time in re-planning. It also used heuristic search methods similar to ‘A* search’ and was able to incrementally learn information from its search to re-plan an alternative minimum cost path to the goal when new obstacles were encountered. The main feature of D* lite that made it more feasible than A* search was its ability to partially re-plan the path, instead of the entire path, saving precious computation time during real time applications. It was also simple to manipulate and integrate into the overall navigation system created in this project, making it preferred to the more complex planning algorithms.

2.4 Trajectory Control Techniques

When autonomous robots are considered, a key requirement is the ability to move from one point to another by considering the available methods of motion such as wheels / legs / belts as well as the surrounding environment such as the terrain and if there are any obstacles present. Simple movements can be carried out without complex methods, but in areas where the terrain is ever changing and new hazards can enter the environment, autonomous mobile robots that are equipped with the capability of learning and adapting to new situations are a necessity. The trajectory control system is the factor that decides on the performance of this ability of a robot. Generally, a trajectory controller tries to follow a given path with the least possible error. The performance of the controller usually directly determines the range of application of the robot as well as the reliability of the whole control system.

For the purpose of this project, work was focused on controlling a 4-wheel differentially driven skid steer mobile robot designed for navigating in an oil palm plantation environment. Although this steering scheme yielded some mechanical advantages such as increased structural robustness and the elimination of a steering mechanism, the motion control was a challenge. To achieve proper steering along a curved path, all of the wheels needed to skid laterally. As a result of skidding, the robot tended to lose motion stability when the centre of rotation became too big. Furthermore, considering the environment of application, factors such as ground force interactions, varying terrain, wheel slip and sensor inaccuracies required instant reactions on the field and not all conditions could be pre-programmed by the user. As a result, a controller that was able to eliminate noise and learn while in the field was necessary. In certain cases, such as [95], by referring to the mechanics of the robot, controllers for the kinematic and dynamic aspects could be created, which were then linked to create an overall control structure for the robot. Though the computation of the kinematics and dynamics are essential for most systems, other additional types of control are commonly integrated into the system to improve their functionality and performance.

A popular technique used for trajectory tracking is based on the use of the Kalman Filter which was created by R. E. Kalman in 1960 [96]. The method used a variable known

as the Kalman Gain, which enabled the calculation of an expected output, based on the measurement from sensors and a prediction made via calculation of the system model. Due to the predictive nature of Kalman Filters, they are usually used either for predicting the state of the current object based on data with errors, or to track the next location / trajectories of an object to follow. Chen et al [97] used the Kalman Filter to predict the linear and angular velocity of a moving object, which was then used to control a robot to follow the object. Koc et al [98] used the filter to determine trajectories of table tennis balls for the control of a table tennis robot. The system was able to predict the balls speed and motion, which resulted in the robot being controlled to either defend or attack based on the arriving pattern. Ngatini et al [99] made use of variants called the Ensemble Kalman Filter (EnKF) and Fuzzy Kalman Filter (FKF) to estimate the position of an autonomous underwater vehicle (AUV), which was then used to estimate the trajectory of the AUV. Results showed that the EnKF performed much better than the FKF, with higher estimation accuracies and faster computation times.

Moving onto other types of control, Proportional-Integral-Derivative (PID) control was widely used in the past for its linearity and robustness in handling data. The kinematic and dynamic models of the robot had to be designed correctly for accurate application [100]. Caracciolo et al. [101] used a dynamic model to design the motion control, instead of a simpler kinematic model. The resulting control design was able to eliminate unwanted lateral skidding enabling the robot to stick to the required path even in uneven terrain conditions. Kozlowski and Pazderski [102] developed a kinematic controller to improve on the results in [101], to overcome the issues with regulation. The controller was more stable and robust to dynamic parameters of uncertainty. Ouyang et al. [103] used a PID control scheme for tracking control in a position domain problem using contour error in addition to kinematic parameters. It was found that such a controller adopted the master-slave synchronised control concept which in turn uses cross-coupled control and event-driven control depending on applications, resulting in better performance. Hoffman et al [104] tried to implement a PID controller in quad-rotor helicopter trajectory tracking control. The system relied on single optimization and had to make a compromise between complexity of the planned path as well as the rate of update of the control input. Aggressive flights could typically fly up to 15° of bank angle and the controller was able to perform well up to its programmed limit of 30°

without apparent degradation. The drawback of PID method is that it is a feedback control system with fixed parameters, lacking the flexibility and adaptability [100]. This implies that a PID controller which lacks the ability to adapt is unsuitable for varied environments.

Meanwhile, Fuzzy Logic, like any other artificial intelligence (AI) methods, was created with an objective of developing a computational method which is able to perform reasoning and problem solving tasks which require human intelligence [105]. Fuzzy Logic also tries to explore a trade-off between precision and the cost in developing an approximate model of a complex system or function. Zhang et al. [106] used a fuzzy logic cross coupling controller for minimizing the orientation error of a differential drive robot, performing better than a similar controller that used PID instead. Chwa [107] used fuzzy adaptive tracking control to handle both kinematic and dynamic disturbances. The system was more capable of tackling uncertain errors compared to non-adaptive controllers of the same type. Kayacan at al [27] used a type 2 fuzzy neural network along with a PD controller for the trajectory control of an autonomous tractor and Begnini et al. [108] used adaptive fuzzy control for the kinematic and PD control for the dynamic control of a differential wheeled robot. Both resulting systems were robust as well as computationally simple, providing better performance than individual schemes. Ahmad et al [109] proposed a Fuzzy-Proportional Derivative (Fuzzy-PD) type of controller for an application in a steering motion control scheme of two-wheeled wheelchairs in confined spaces. In this research, a cross-compensator of proportional action was introduced and the percentage of steady state error in the system response was decreased to 5% and it settled in 2 seconds.

Benmakhlouf et al [16] proposed a path tracking controller for differential driven mobile robots based on Fuzzy Logic to mimic a human driver experience. It used a simple architecture (2 fuzzy modules) based on a human driver driving experience. Together with a new measure of look-ahead curvature (LAC), the proposed controller drove the robot through a trajectory formed by discrete waypoints with high speed and precision. Although Fuzzy Logic controllers can provide better performance compared to traditional control systems, they require longer design time, starting from initial design to subsequent tuning in order to achieve good performance [110]. When the

operating conditions are varied significantly from the design conditions, the controller is unable to demonstrate robust performance.

On the other hand, Neural Networks offer benefits such as the ability to deal with nonlinearity [111]. Neural Network controllers possesses the ability to adapt to a changing working environment even though the working environment could be greatly different from design conditions [112]. Jiang [113] proposed the use of a traditional PID controller working in parallel with a neural network controller in trajectory control of a manipulator. The presence of the neural network was to approximate the dynamics and inverse dynamics in order to compensate the dynamic effects of the robot. The results showed that after some time the trajectory tracking accuracy remained consistent, given that the working environment had not changed. Xiao et al. [114] presented an iterative learning control scheme comprising of a Neural Network controller and a linear feedback controller, suitable for real-time implementation. They proposed a robust weight-tuning algorithm for multi-layer Neural Network using the dead-zone technique to approximate the inverse dynamics of the robot. A fixed gain PD controller stabilised the system along a desired trajectory while the Neural Network controller predicted the desired actuator torque. The result of the study showed that the neural network controller was capable of reducing tracking errors after several iterations and was less sensitive to noise.

In general, there are three types of learning modes for neural networks, namely supervised, semi-supervised and unsupervised learning methods, while the networks can be either adaptive or non-adaptive. Learning methods could also be separated as offline, where the network is trained when the system is not running, and online learning when learning and system functionality are done simultaneously. The learning method adopted by a neural network algorithm defines the working operation of the network itself. For instance, some networks do not require prior knowledge of the training data while others could adapt to varying working environments. Therefore, the ability of a network to eliminate or to reduce noise present in the training data is very important for most of the everyday applications. In view of the common Neural Network architecture, most of the Neural Networks require certain information regarding the input data space to be known beforehand [115]. This information could

be the statistical distribution of data such as covariance or simply the learning environment of the networks.

Yang and Meng [116] proposed a real time motion planning system with obstacle avoidance using neural networks for both a mobile robot and a multi joint manipulator. The system was able to deal with a dynamic environment with no prior information regarding the environment. Seyr et al [117] used a simple offline trained single layer feed-forward perceptron network for the predictive control of a two-wheeled mobile robot. The integrated control error was used to adjust control variables of the robot. Yanfeng et al [118] used a back stepping approach with a neural network for controlling a mobile robot for welding curved seams. The system was able to function with high performance without the use of a dynamic model of the robot. Though the use of offline trained networks requires no post-training of the network after implementation and therefore are simple, they are unable to react to new data / situations that are outside the original training data.

Unlabelled data are abundant but disorganised, and can be learned via unsupervised learning techniques, without prior knowledge of the data being learnt. Such networks are used to model the structure of samples in the measurement or attribute space, regardless of the probability density function or in terms of clustering. The learning strategy could be either self-organizing learning or statistical learning. Fierro and Lewis [119] used a neural network dynamic controller designed by using a feed-forward neural network along with a designed kinematic controller to perform trajectory tracking and path following on a non-holonomic three wheeled robot. The weight update procedure of the network was changed to enable the weights to be updated on-line by learning the dynamics of the robot. Bugeja et al [120] used dual adaptive neural control schemes via the use of Gaussian radial basis function and multilayer perceptron networks to prove the better functionality of adaptive networks. Though these two networks were non-adaptive in their original states, they were transformed by the use of functions that estimated the current output, enabling the user to skip the offline pre-training step and directly move onto online training and control.

Xu et al [121] used a neural network based sliding mode controller for trajectory control of a wheeled mobile manipulator to overcome the shortcomings of the sliding mode

controller with regards to noise amplification and complexity. Rossomando et al. [122] used an adaptive radial basis function neural network for dynamic compensation of the uncertainty of a nominal dynamics model. Li et al [123] proposed a model predictive control scheme that incorporated primal dual neural network optimization with unsupervised learning for the trajectory tracking of a non-holonomic robot. Yen et al [124] also used an adaptive neural network controller with dynamic feedback for electrically driven non-holonomic systems to reduce the tracking error to a minimum. Hoang and Kang [17] used neural network based adaptive tracking control for dealing with wheel slip and external disturbances. The online learning method assisted in the reduction of tracking errors. Cui et al. [18] used two neural networks to deal with unknown disturbances and parameter uncertainties of an autonomous underwater vehicle, and proved that the proposed method had a much higher performance as compared to PD control. An adaptive neural network controller was used by Li et al. [125] for controlling unknown dynamics and motion constraints of robotic manipulators and Sun et al. [19] used a RBF network to approximate the unknown model of a biped robot.

A neural network that was capable of both unsupervised learning as well as adaptive / incremental learning was created by Hasegawa Laboratory. Named the Self-Organised Incremental Neural Network (SOINN), it was built and developed based on the Self-Organising Map (SOM) [126], and possessed the capability of associating, reasoning, knowledge-transferring and forecasting by organising unlabelled data [127]. SOINN, resembling SOM, did not need any information on the input data space for efficient performance. The structure of the SOINN, unlike SOM, was capable of changing according to the learning environment at each epoch of training. In comparison to the adaptive networks mentioned above, SOINN was inherently adaptive in nature and the network structure ensured fast training during incremental learning, which was of great importance when new data had to be included. Though SOINN had been used for map building and navigation purposes by Kawewong et al [128] and Tangruamsub et al [129] respectively, SOINN had not been applied for the purpose of trajectory control as in this project.

As the trajectory controller needed for this project had to be able to provide varied outputs depending on the motions to be carried out, it had to be adaptive as well as a

network that could be trained incrementally. The SOINN variant known as the Enhanced SOINN (ESOINN) [10] was chosen as it had better performance than the original while being able to do everything the original could. The ESOINN had a single layer structure as opposed to the dual layer in SOINN and it separated clusters with high overlap while using lesser parameters and being more stable. The ESOINN was a perfect solution to the trajectory control application, as it could perform the required tasks, and as it was an unsupervised learning algorithm, the nature of this method enabled the creation of a controller that did not require a kinematic model of the robot as it was capable of linking sets of data and matching completely new inputs to suitable outputs. However, since the ESOINN was developed for classification problems, it was only capable of producing discrete outputs, which caused a problem when continuous outputs were necessary as in the case of trajectory control, as the distances to travel and motions to perform were varied depending on the situation and could not be trained in completely.

To overcome the issue of discrete outputs, one method to be used was interpolation. This would enable a network to produce continuous outputs, if the discrete outputs were not applicable in the given situation. If the output of ESOINN was restricted to discrete data only, the controller would be unable to perform satisfactorily. An accumulation of round errors (due to quantized outputs) could be a huge problem with increasing runtime of the system. Yang et al. [130] studied 12 different interpolation methods and the efficiency as well as accuracy of each method was examined. RBF showed the least error though it took up a lot of processing time. On the other hand, Polynomial Regression showed a major discrepancy despite having the shortest processing time. Triangulation with linear interpolation had the lowest processing time and yielded a reasonable accuracy while Nearest Neighbours and Inverse Distance Weighting (IDW) both took a similar amount of time with IDW showing a better result. Aupetit et al. [131] proposed a new multi-dimensional interpolation method for function approximation using SOM. The function approximator produced continuous outputs since it took in continuous inputs while SOM produced discrete outputs. The proposal used the Euclidean distance already computed to do self-organization in the input space. The usage of Local Linear Mapping (LLM) together with this information of Euclidean distance improved the approximation quality of standard SOMs. Kalteh et al also explored the interpolation method in representation of monthly precipitation data using

SOMs [132]. The experiment conducted had shown that SOM with interpolation gave better results when working with homogenous data. This showed that interpolation when used in conjunction with a classifier with discrete outputs was a feasible way to create a system with good performance.

In conclusion, it was deemed that the ESOINN method was to be applied as the trajectory controller for this project based on the advantages and feasibility mentioned above. It was modified by the addition of the IDW interpolation method created by Shepard [11] to provide continuous outputs, leading to the creation of a novel unsupervised adaptive learning algorithm dubbed as IDW-ESOINN. The original ESOINN algorithm is described in the following subsection as a reference, followed by the IDW algorithm. The modified algorithm created for the IDW-ESOINN is described in Section 3.4.

2.4.1 The Enhanced Self-Organising Incremental Learning Neural Network (ESOINN) Algorithm

The algorithm of the original ESOINN is given as follows [10]:

1. Two nodes with weight vectors, W_a are initialised by choosing data from input pattern, X randomly. This set of nodes is denoted as A. Connection set, C is also initialised to the empty set.
2. A new pattern $x \in R^n$ is inputted into the network.
3. Find the closest node, a_1 (also known as the winner node) and the second closest node, a_2 (the second winner) according to the calculation:

$$a_1 = arg\,min_{a \in A} \left|\left| x - W_a \right|\right| \quad (5)$$

$$a_2 = arg\,min_{a \in A[a_1]} \left|\left| x - W_a \right|\right| \quad (6)$$

   If the distance is greater than similarity threshold, T_a, the input data is a new node, add it to A and go to step 2 to process successive data. T is calculated as

$$T_i = \max_{j \in N_i} \left|\left| W_i - W_j \right|\right| \quad (7)$$

   where N refers to neighbouring nodes.

4. The age of all edges linked with winning node a_1 is increased by 1.

5. For building the connection between the nodes:
   a. Connect the two nodes with an edge if the winner or the second winner is a new node, or
   b. Connect the two nodes with an edge if the winner and second winner are both belonged to the same subclass, or
   c. If winner belongs to subclass *K*, the second winner belongs to subclass *L*, combine the two nodes if $\min(h_{winner}, h_{second\ winner}) > \alpha_K K_{max}\ or\ \alpha_L L_{max}$ where h is the density of node and α as a parameter determined by *T*. Otherwise, do not connect both nodes and remove any existing connection between these two nodes.
6. Update the density of the winner using

$$h_i = \frac{1}{N} S_i = \frac{1}{N} \sum_{j=1}^{n} \left(\sum_{m=1}^{\lambda} P_i \right) \quad (8)$$

where S is the sum of points for node i and P being "point" of the node. Add 1 to a local accumulated number of signals M_{a1},

$$M_{a1} = (t+1) = M_{a1}(t) + 1 \quad (9)$$

7. Adapt the weight vectors of the winner and its direct topological neighbours by a fraction $\in_1 (t)$ and $\in_2 (t)$ of the total distance to the input signal,

$$\Delta W_{a1} = \in_1 (M_{a1})(\xi - W_{a1}) \quad (10)$$

$$\Delta W_i = \in_2 (M_{a1})(\xi - W_i) \quad (11)$$

8. Find the edge whose ages are greater than a predefined parameter age_{max}, then remove such edges.
9. If the number of input signals generated so far is an integer multiple of parameter λ,
   a. Update the subclass label of every node.
   b. Delete nodes resulting from noise as follow:
      i. For all nodes in A, if node A has two neighbours, and $h_a < c_1 \sum_{j=1}^{N_A} \frac{h_j}{N_A}$, then remove the node a. N_A is the number of nodes in node set A.
      ii. For all nodes in A, if node A has one neighbour and $h_a < c_2 \sum_{j=1}^{N_A} \frac{h_j}{N_A}$, then remove the node A.

iii. For all nodes in A, if node A has no neighbour, then remove node a.

10. If the learning process is finished, classify nodes to different classes, then report the number of classes, output the prototype vectors of every class, and stop the learning process.
11. Go to Step (2) to continue unsupervised online learning if the learning is not finished.

A point to note is that for incremental learning, the algorithm works in exactly the same way, from steps 2-11, except that the initial conditions for the involved parameters are already set based on the previously trained network. The algorithm makes use of the previous parameters to add in the new data points to the current network and updates the current nodes by either moving them if new data are added or removing them if found to be noisy or redundant. A flowchart of the simplified training process is shown in Fig. 2.4.

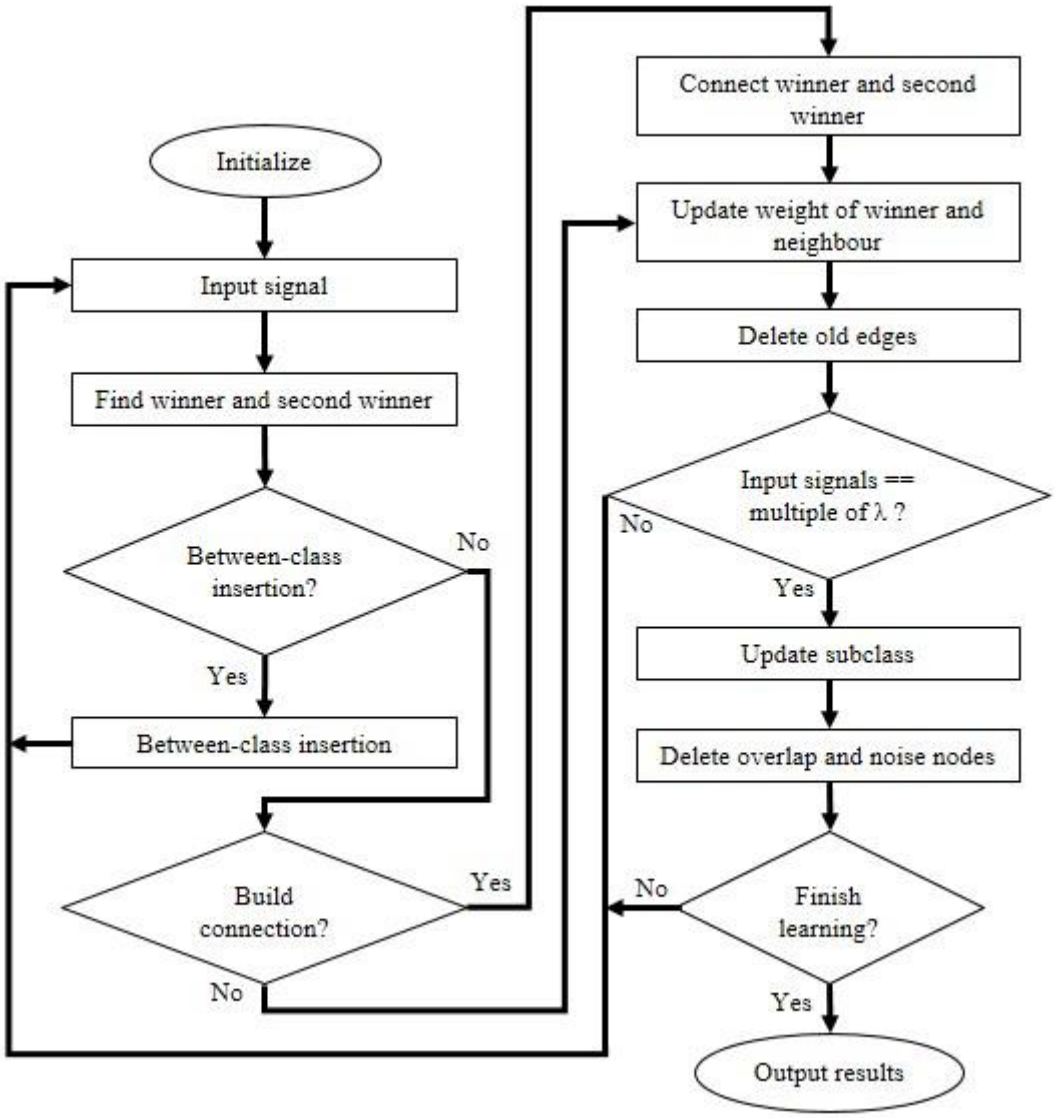


Fig. 2.4 - Flowchart of ESOINN [10]

2.4.2 The Inverse Distance Weighting (IDW) interpolation algorithm

The IDW interpolation algorithm [11] works by interpolating the value of an unknown point, by calculating a weighted average using the values of known points in a scattered set of points. The algorithm works as follows:

1. Calculate the 'sum of ratio' by dividing the value *z* of each of the known points *i = 1-k*, where *k* is the total number of known points, by the distance *d* of the unknown point to the known point.

$$Sum\ of\ ratio\ (SR) = \sum_{i=1}^{k} \left(\frac{z_i}{d_i}\right) \quad (12)$$

2. Calculate the 'reciprocal of powered distances' by raising the power of each distance *d* by the power parameter *p*, with greater values of *p* assigning greater influence to values closer to the unknown point and vice-versa. The reciprocal for all the known points is summed together.

$$Reciprocal\ of\ powered\ distance\ (RP) = \sum_{i=1}^{k} \left(\frac{1}{d_i{}^p}\right) \quad (13)$$

3. The value of the unknown point *X* is then found by using the relation:

$$Value\ of\ X = \frac{SR}{RP} = \frac{\sum_{i=1}^{k}\left(\frac{z_i}{d_i}\right)}{\sum_{i=1}^{k}\left(\frac{1}{d_i{}^p}\right)} \quad (14)$$

2.5 Summary of Literature Review

The cited literature in this chapter enabled the consideration of the various methods currently available, so that a clear decision could be made with regards to the methods used in this project, for the creation of the various parts of the navigation system.

There have been numerous work done in creating machines for agriculture, both human driven and semi / fully automated, yet there is a lack of automation when oil palm plantations are considered. Given the nature of the plantation environment which is unstructured and unpredictable as it is fully outdoors, a smart system is required to accomplish any given tasks autonomously. Based on the literature, a machine for this application has to be robust and manoeuvrable to handle the terrain, and mounted with the necessary amount of sensors for inputs, typically a camera and distance measurement sensors. The ever changing plantation environment eliminated the use of fixed tracks / paths unlike those projects situated in greenhouses or cleanly maintained orchards. Ideally a four wheeled / track driven skid-steer robot would suit the purpose, with a size that is just large enough to accomplish the task at hand.

The navigation system for an automated robot, requires an object detection system, mainly centred around the key object of interest, which in this case was the oil palm tree. Detection would take the input from either a camera or depth sensor or both, which would then be pre-processed / features extracted then fed into a detector for identification of the trees. The detector should have a high accuracy, while being able to isolate the tree clearly from the background. As the oil palm plantation environment differed to the other works mentioned in the literature that had done tree detection, due to the low contrast background and foliage on the tree trunks, a unique method had to be created for this project. A novel pre-processing method was created to remove the ground area and a cascade type detector – Viola and Jones detector – was chosen to detect the trees, mainly chosen for its fast processing speed. To increase the accuracy of detections, depth data was also used along with colour images, similar to other methods used in the literature, via the use of the KINECT sensor.

For a robot to travel to the key object of interest, a path has to be planned. This path should be the most efficient one, taking the shortest route possible, while avoiding all the obstacles along the way. To ensure the obstacles are detected, sufficient sensors to detect the obstacles and the distance to them are required, resulting in the choice of ultrasonic sensors to be used for this purpose. For the path planning process, an algorithm that was computationally light and could re-plan paths on the go was desired, and based on the literature, the 'moving target D* lite' algorithm was chosen for this purpose. This was preferred over other methods such as A* search and the Dijkstra algorithms, which were more inefficient in comparison.

For following the calculated path, a reliable controller was required that would output the motions of the robot to be performed accurately. Given the unstructured nature of the plantation, a system that would be able to adapt to new situations without human interference was highly desirable. Based on the cited works, an incremental learning type controller would be highly suitable as it would be able to learn new motions while operating in the field. The ESOINN was capable of this, while being an unsupervised learning algorithm, making it possible for the network to form relationships between paths and motions by itself. This made it preferable when compared to traditional PID / fuzzy controllers that weren't able to adapt to new unknown inputs. To increase the range of possible motions, an interpolation method known as IDW was added to convert the discrete outputs of the ESOINN to continuous outputs, thereby increasing the resolution of the network.

CHAPTER 3: METHODOLOGY

This chapter contains 4 sub sections, with the focus being on the methods used to achieve the objectives of this project. The first section is on the robot rover created for the project for the purpose of testing the navigation system. The next section focuses on the object detection method used for identification of the oil palm trees via tree trunk detection, followed by the third section which focuses on the path planning method used to calculate the path the robot should take to reach the detected trees. The final section contains the methods used to train and implement the trajectory controller that is used to control the motion of the robot so that it may follow the given path.

3.1 Robot Rover

The robot rover was designed specifically for the testing of the navigation system that was created for this project. The rover had to be functional as a prototype so that all the necessary sensors and hardware could be mounted and that it could successfully navigate the terrain of an oil palm plantation. The plantation environment had grassy / slightly muddy terrain, with the weather ranging from dry to heavy rainfalls, which affected ground conditions. The robot would have to traverse across small puddles of water, tree roots, small shrubbery as well as sloped areas. This emphasised the need for a robust design, with high manoeuvrability.

3.1.1 Chassis Design

When designing the chassis, a few assumptions were made: the maximum navigable slope would be 30^{o} and the plantation areas the robot would be applied to would be under decent maintenance, so that no large obstacles / sudden high drops were present. This resulted in the elimination of a need for a proper suspension system or articulated chassis, as the robot would be able to be used for this project's purposes without them.

Taking the mentioned factors into account, the design choices made are shown in Table 3.1.

Table 3.1 - Robot Design Decisions

Category	Design Choice	Design Details
Shape and size	Should be able to contain all necessary components as well as a laptop used for processing Basic box shape was sufficient and was a stable configuration The inside should be waterproof to prevent water damage	Robot chassis designed with dimensions of 820mm (l) x 500mm (w) x 260mm (h) Metal cuboid base frame with plates attached on the 6 sides as covers to seal the inside. Covers attached with bolts and nuts for easy assembly, de-assembly and modifications.
Strength	Frame needed as a base for supporting the structure of robot	
Motion	A four-wheel drive layout was decided upon due to its stability and manoeuvrability Wheels with good grip were needed for the terrain Ground clearance to reduce tangling at the bottom	Rubber tread wheels with a diameter of 18cm were chosen. Ground clearance of 10cm was achieved through the chosen wheels.
Materials	Rigid beam-like material needed for box frame Hardy material needed for base and sides All materials needed to be corrosion-resistant	Aluminium profile was used for the robot frame and 3mm thick sheets were chosen as a base plate for its strength Acrylic sheets of 5mm thickness were chosen to form the rest of the sides and the top due to its light weight Mild-steel sheets of 3mm thickness were chosen to form motor housing due to their strength.

The total weight of the robot, including the components chosen came to approximately 55kg.

A few points to note are:

- A laptop was chosen as the main processor instead of other available micro-processors as this would allow the testing of varied systems with a higher limit on the processes that can be carried out.

- A triangular shape at the front and back of the robot were designed to help with the motion as well as increased stability. When the robot was travelling up an incline or over a bump that was slightly higher than the ground clearance, the triangular surface enabled the robot to bypass this obstacle, given that the overall height to overcome was within limits.

- The mount at the top of the robot was designed for housing the camera. The cylinder was mild-steel with an acrylic box to house the camera. The length was decided so that the view obtained would be at a reasonable level for navigation tasks.

3.1.2 Component Selection

Given the necessary requirements of the robot, the appropriate devices needed as well as the available types and the final choices made are mentioned below.

(A) Sensors for Distance Measurement:

The sensor chosen was the Ultrasonic Ranging Module SN-HC-SR04, with a working range of 4 meters and an angle of measurement of 30°. This was chosen over the GRV-ULTRASONIC and SN-LV-EZ1 sensors as it had a lower cost and had suitable operating parameters. As the sensor's detection was limited to only a linear measurement of the direction it was facing, a total of 5 sensors were chosen as an appropriate number to get a better reading of the area around the robot.

(B) Sensor for Movement Measurement:

A compass was needed for direction measurement and an encoder was needed as well for measuring the output of the motors. An inertial measurement unit (IMU) was chosen as it had a combination of a digital compass, accelerometer as well as a gyroscope. The

model chosen was 'MinIMU-9 v3', as it was a low cost option that had fulfilled the necessary requirements of the project. An encoder was not needed as the selected rear motors came with encoders.

(C) Sensor for Position Location:

For accurate positioning of the robot, the current location needed to be known. For this application, the GPS module 'SKM53 GPS' was chosen to be used instead of the 'GT-U7' model, given its lower cost and acceptable range of function.

(D) Camera for Vision System:

To act as an input for the vision system, a camera had to be included. The Microsoft XBOX KINECT sensor was used to function as the vision system, as it contained both an RGB camera in addition to an infra-red depth sensor that could be used to obtain 3D depth images. It was also considered to be a low cost alternative to the regular LIDAR sensors. It had a field of view of 57 degrees horizontally and 43 degrees vertically, with a resolution of 640*480 pixels and 30fps capture rate. The KINECT was mounted at a height of 65cm above ground and was angled to be parallel to the ground.

(E) Motors for Motion:

Given that the weight of the robot was estimated to be 55kg and a four-wheel drive setting chosen as the motion method, 4 motors were needed to operate the robot, with one motor per wheel. According to the motor torque calculations, the required torque per wheel was 12.13Nm and the servo motor 'IG-71HM' was chosen as it provided a torque of 12.25Nm.

(F) Controller for Components:

A micro-controller was needed as a means of obtaining inputs from the components and sending output signals from the laptop. The Arduino Mega 2560 was chosen as the most appropriate due its reasonable cost, higher number of input/output pins as compared to the Arduino UNO and easy compatibility with most of the chosen components as well as its easy interfacing with Matlab as compared to a Raspberry Pie.

(G) Main Processor

The main processing was done by a laptop mounted inside the robot rover. The laptop used an Asus GL552VX with an Intel Core I7-6700 CPU with a processing speed of 2.6GHz and 8GB of usable RAM.

(H) Power Source:

A source was needed to supply power to all the components, and to deem the necessary parameters of the source the power requirements of the components had to be considered. Table 3.2 below and the calculations that follow enabled the choice of the power source.

Table 3.2 - Current consumption of components

Component	No. of units	Current drawn	Total current
Motors	4	1500mA	6000mA
Ultrasonic sensors	5	15mA	75mA
IMU	1	10mA	10mA
GPS	1	20mA	20mA
Total			6105mA

As the motors drew a much higher current than any of the other components, it was deemed that a separate source for the motors was needed to prevent component damage. A 12V source was chosen to power the motors and an 11.1V source to power the rest.

The calculations below show the time limit of each source's functionality:

$$11.1V\ battery\ charge\ \rightarrow \frac{2200mAh}{105mA} = \ 20.9\ hours$$

$$12V\ battery\ charge\ \rightarrow \frac{14Ah}{6A} = 2.3\ hours$$

Finally, one 11.1V lithium ion battery and four 12V lead-acid batteries were chosen to power the robot's motors and sensors. An additional 11.1V battery was purchased later to power the KINECT sensor by itself, as it required a higher voltage level as compared to the other sensors which only needed 5V which was obtained by stepping down the first 11.1V battery.

3.1.3 Component Layout

The internal space of the robot was divided into 3 layers.

- Layer 1 (bottom) contained the 4 12V batteries, 4 motors and 2 encoders
- Layer 2 (middle) contained the Arduino Mega, GPS module, motor drivers and the 11.1V battery
- Layer 3 (top) was the space assigned for the laptop

The KINECT was mounted at the bar mounted at the top of the robot and the IMU was placed there as well to avoid interference from the other components due to its sensitivity to magnetic fields. The overall connections made among the components are displayed through the flowchart in Fig. 3.1 below:

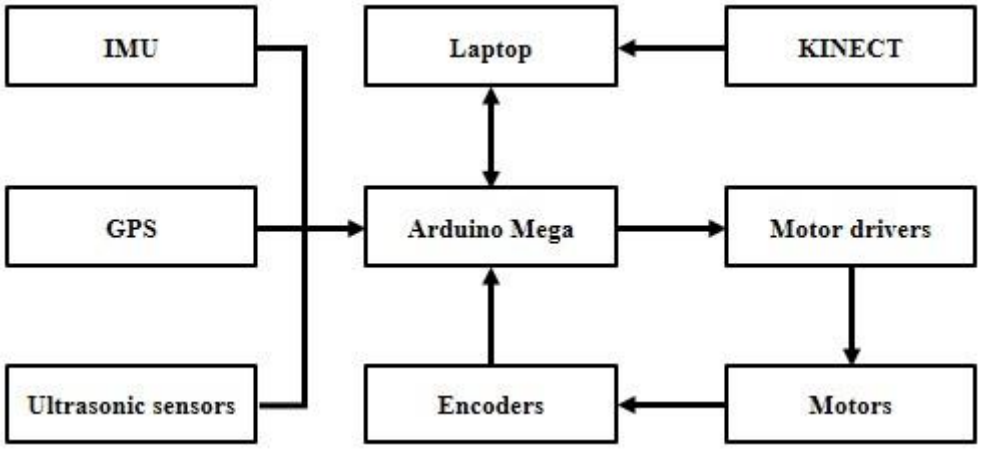


Fig. 3.1 - Robot component layout

3.1.4 Fully Constructed Robot Rover

The fully constructed robot was a four-wheeled, differentially driven, skid-steering robot. One of the unique features that lead to its robustness was its steering scheme. With each side of wheel pairs driven at a different rotational velocity, the wheels of the robot skid laterally to change its direction of travel, while in a conventional vehicle, no skidding is assumed during directional changing. The robot was able to manoeuvre with ease in confined spaces and rough terrains which was required in an oil palm plantation. The ability to move around obstacles and in between trees required a high level of manoeuvring that skid steering could provide. This configuration also made the mechanical structure of the robot to be more rigid, due to lesser moving parts. Without the need of a steering column, the overall construction of the robot was also made simpler. This robot also required minimal maintenance work, as the only moving parts that would wear off were the wheels and axles. The robot after completion of the construction and assembly is shown in Fig. 3.2 with the circles in red showing the areas where the ultrasonic sensors were mounted. The mounts were temporarily created by using cardboard, so that the orientation of the sensors could be changed during the testing process of the navigation system as required.

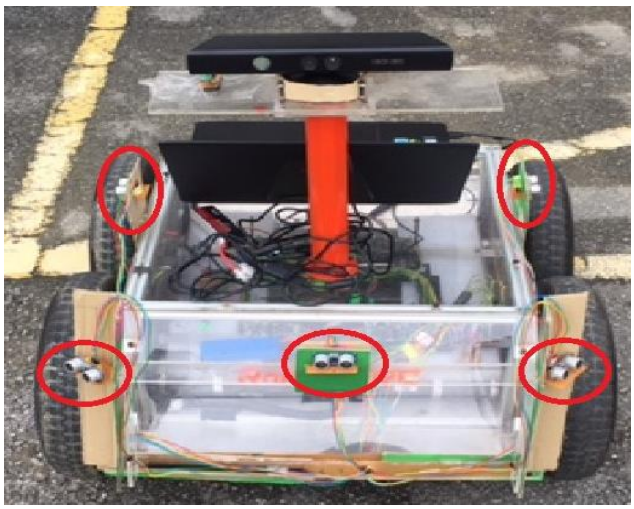


Fig. 3.2 - Fully constructed robot rover

3.2 Tree Trunk Detection System

As the focus of this project was the navigation system, an exact end task such as weeding or harvesting was not assigned to the robot. To this effect, it was decided that the robot should be able to move from tree to tree within the plantation plots, as this motion would function as the base for tasks such as fruit picking and chemical application. Tree trunks were to be used as key waypoints while obstacles to be encountered would be branches, fresh fruit bunches and shrubbery. The main focus was to be on the detection of the tree trunks as this would enable work to progress on to the other stages of the system.

The detection method can be split into two paths: colour image processing and depth data processing. The colour image was pre-processed to eliminate the ground area that appears in an image and was then fed into the Viola and Jones detector. The depth data was converted to a black and white image, then processed via the Hough transform for detecting the lines that made up the sides of the trunk. All detections made were then combined to output the locations of the trunks present in view, after which the nearest tree to the robot was confirmed and the distance to it was estimated. A flowchart displaying the detection method is shown in Fig. 3.3 and they are further explained in the following subsections. The created tree detection algorithm was dubbed as the Oil Palm Tree Detector (OPTD) and will be referred to as such in the remainder of this thesis.

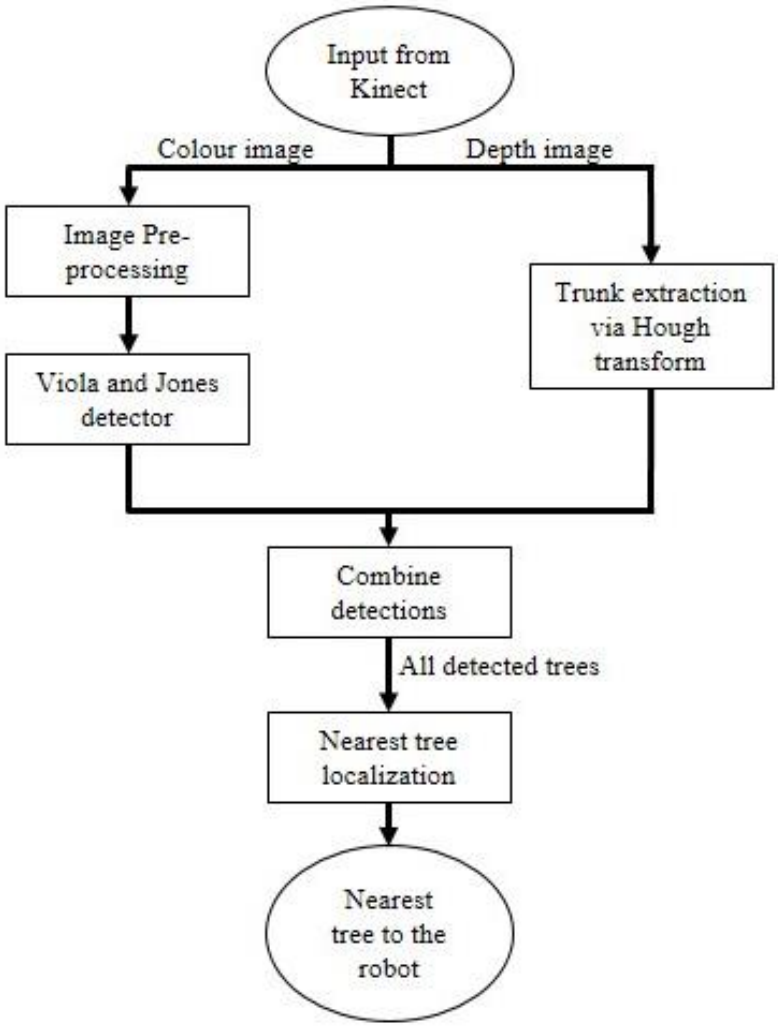


Fig. 3.3 - Flowchart of the OPTD method

3.2.1 Data Collection

Video data was collected by driving the robot through various areas of the plantation via remote control and recording the environment through the KINECT sensor. Frames were extracted from the videos to provide images for both training and testing purposes. A sample of the tree trunk images used for training is shown in Fig. 3.4. The depth data did not require high level processing and as a result were excluded from the training process.



Fig. 3.4 - Tree trunk samples for training

As seen in the images, the tree trunks did not have a consistent appearance as varying degrees of growth of parasitic plants were present on most of the trunks. The results from running the detection system in both static images and on real time video streams is shown and discussed in Chapter 4.

3.2.2 Colour Image Processing

The colour image processing section of the OPTD method made use of the RGB colour images extracted from the video stream obtained from the KINECT sensor. The images were pre-processed and then run through a detector to find the tree trunks present in the image.

(A) Novel Pre-processing Method

Due to the height and angle of the KINECT and the distance between the trees, most images contained the plantation floor, to a maximum of half the height of the image depending on the distance to the trees. This caused interference as it gave higher chances for the detector to mistake areas of dark soil as trunks. The angle of the KINECT could not be changed, as having it too low would decrease the view of the trunks and having it higher would cause more light to interfere and make detections more difficult as the tree trunks would be narrower higher up, making it harder to differentiate them from the background as the branches would appear as well. A novel pre-processing method was proposed to eliminate the ground area which was not needed for detection. By doing so, the accuracy of the detector was increased by lowering the number of false alarms through the reduction of the area of the image that was fed into the detector.

The ground elimination from images was done by observing the contrast of the tree trunks to the ground in various colour spaces (RGB, YCBCR, YIQ, HSV and L*a*b). It was observed that the best contrast obtained was in the Hue value of the HSV colour space. To further improve the contrast ratio, different combinations of colour space matrices were checked and the results showed that the highest contrast obtained was when the pixel's Hue (H) value was subtracted from the Blue (B) value from RGB. It was observed that areas where tree trunks or ground with dark soil appeared had small values that were close to black, while leaves on the trees and areas with ground covered with vegetation or light coloured soil appeared much lighter in comparison. Since most of the floor within the square plots of the plantation was covered with vegetation, the method provided greater contrast than a single colour space.

The method was further improved so that when comparing the H and B values, those pixels where B was higher were marked as 1 and those where H was higher were marked as 0. This resulted in a binary image that was easier to manipulate into eliminating areas of an image. The ground was then eliminated by searching for the rows from the top of the image that had a high number (70% of the pixels of a row) of white pixels (light ground and vegetation) and cropping the image to rid the remaining rows from the image. This edited image was then sent into the detector instead of the full image. The image conversion process is visually displayed in Fig. 3.5.

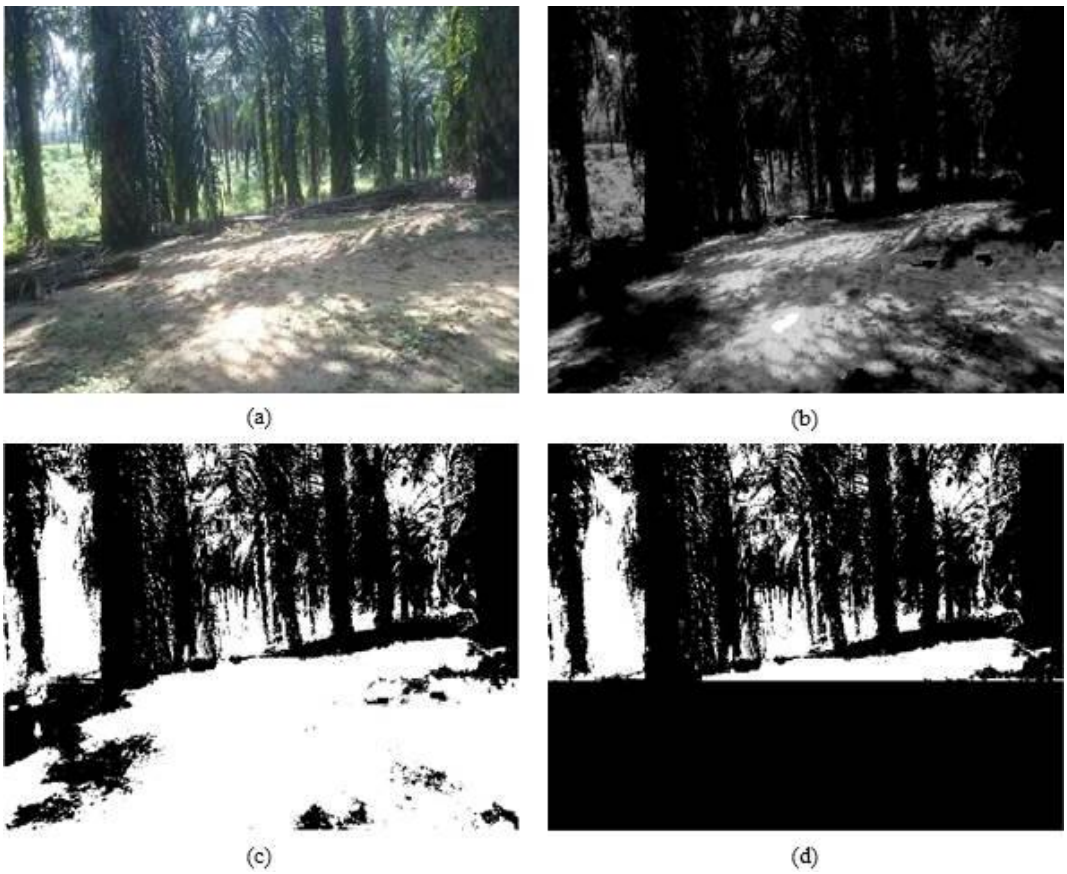


Fig. 3.5 - Image conversion process via proposed pre-processing method: (a) original image; (b) result of B-H; (c) binary transformation of B-H; (d) cropped output image

As seen in Fig. 3.5, the original image was cropped to eliminate the area of the image that did not contain tree trunks. Those rows that are completely black in image (d) are removed from the original and then the resulting image is fed into the detector for further processing.

(B) Cascaded Tree Trunk Detector

The detection system used the Viola and Jones detector as the base detector. The Viola and Jones boosted cascade detector was used in its original state with the use of Haar filters and the Adaboost learning algorithm. After training several detectors, the optimum parameters for the detector with the best performance were: 10 cascade stages and an input sample window size of 119*32pixels. Once the input image went through pre-processing and the ground was eliminated, the grayscale version of the image was then fed into the cascade detector for identification of the tree trunks.

The simplified flowchart displaying the steps involved in the colour processing section of the OPTD method is shown in Fig. 3.6. The first half of the flowchart describes the processes with which the ground was eliminated from the images. The second half shows the steps that were followed as part of the Viola and Jones detector.

The detector created multiple stages, each with a multitude of Haar filters to form a cascade. During training, the filters were chosen so that each stage had a set detection and false positive rate, which gave each stage its own threshold value via the computation of the pixel values by the filters. When a sub-window was input to the detector, for each stage, the filters were applied to the pixels of that window and the output value was compared to the threshold value of the stage. If the value was acceptable, then the window was processed by the next stage and if not, it was rejected and the next sub-window was input to the detector. A sub-window was deemed to be a positive detection only once it had passed through all the stages of the cascade.

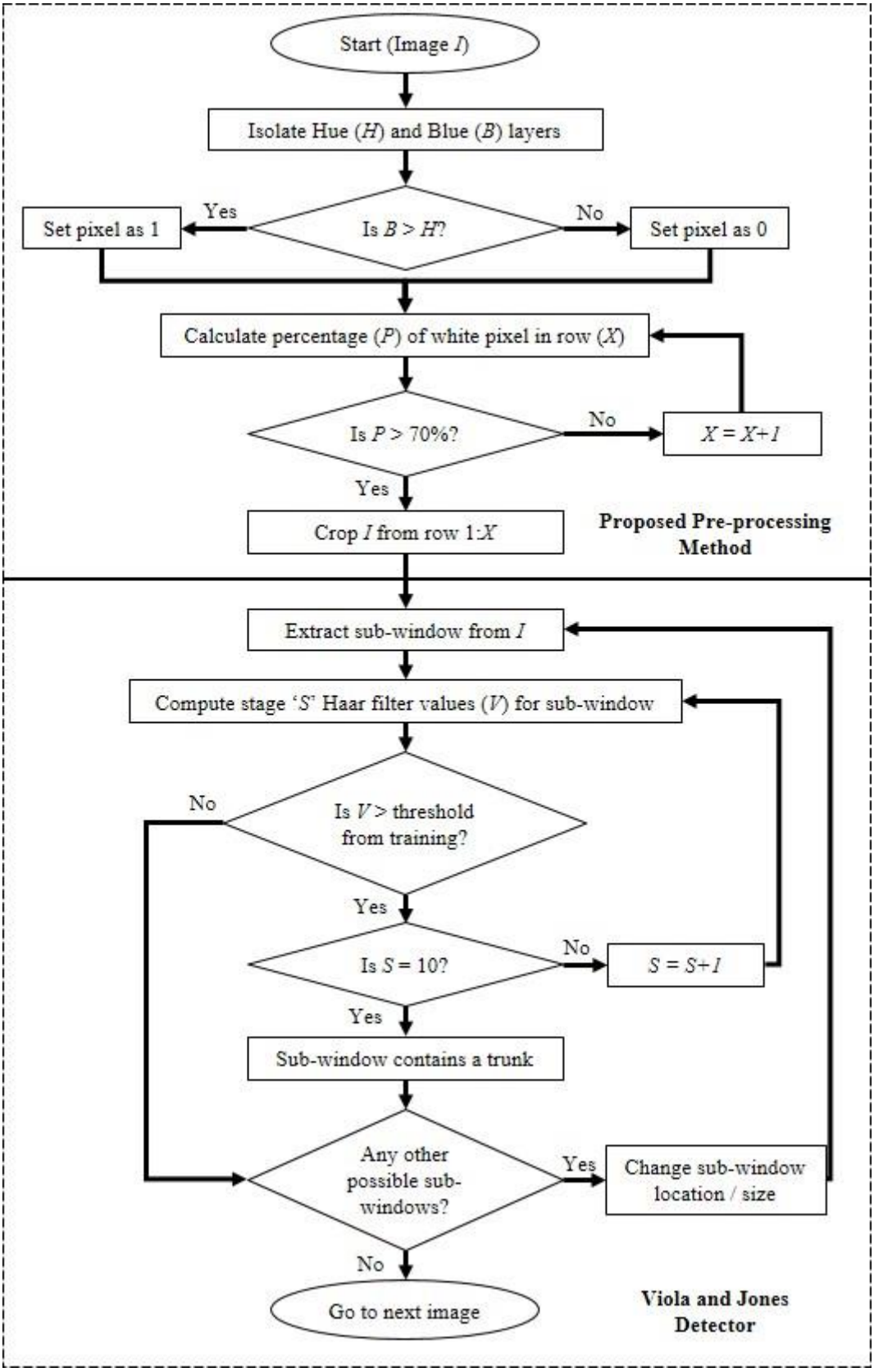


Fig. 3.6 - Flowchart of the colour image processing process

3.2.3 Depth Data Processing

To ensure that the detections made would include trees that were within the range of the depth sensor, the depth data obtained from the KINECT was processed to extract tree trunks as well. Depth data was essential for situations with no or low levels of light where visual information was either non-existent or insufficient to make accurate detections. From indoor tests under regular lighting and outdoor tests at night, the KINECT sensor had a detection range of up to 4 meters, but in an outdoor setting during the day, the range was limited to a maximum of 2 meters. Data was collected during different times of the day and it was found that the depth information was almost non-existent during peak daylight hours. Reliable depth data could only be obtained at dawn, dusk, areas with high shade or at night time.

After analysing recorded data, the following observation was made with regards to the depth data: only 1 tree trunk would appear in a frame, if the trunk was within range of the sensor. This was a result of the distance between tree trunks being larger than the working range of the depth sensor. As only 1 trunk would appear in a frame, high level processing was not needed to detect trees in the data. Instead, the Hough Transform was used to extract line segments from an edge image to obtain the location of the sides of the trunk. The processes carried out on the depth data / image by the OPTD method can be seen in Fig. 3.7.

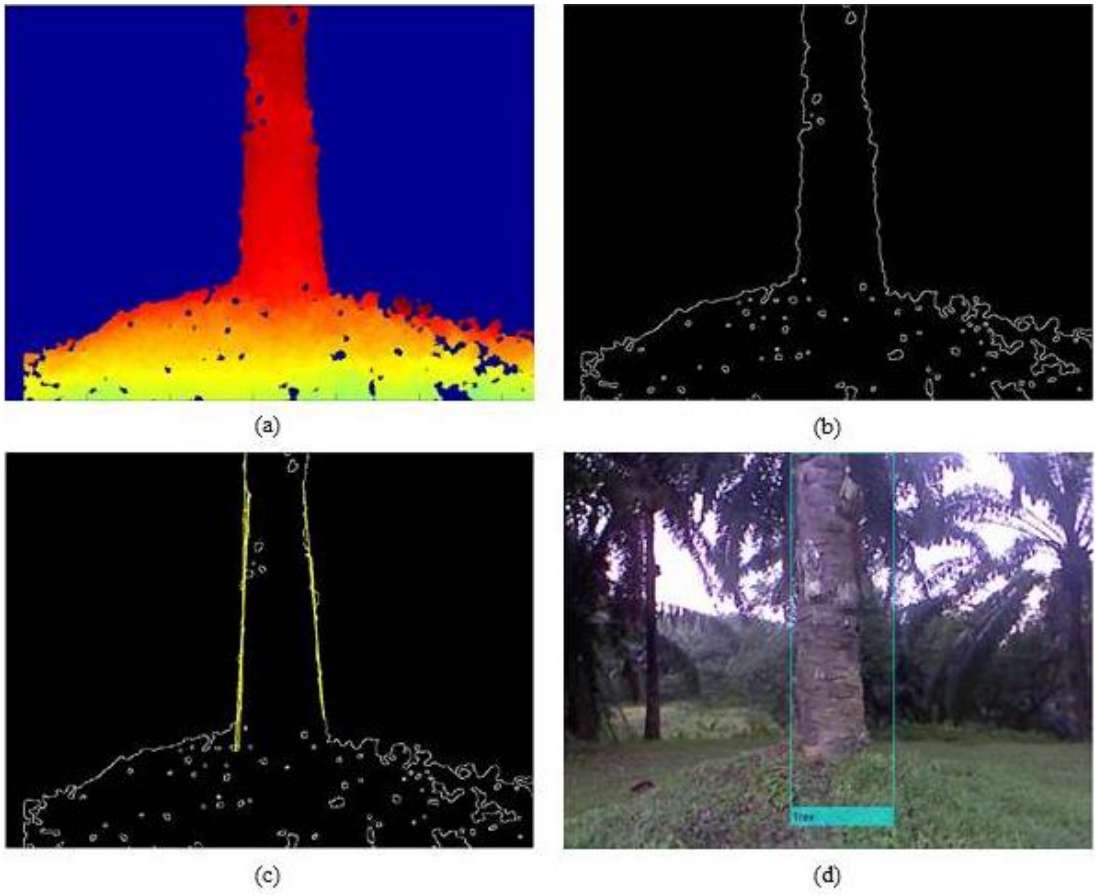


Fig. 3.7 - Depth data conversion process: (a) original depth data; (b) edge detection; (c) line extraction by Hough Transform; (d) detected tree trunk shown in colour image

The depth data was first converted to binary by simply converting pixels without readings to 0 and pixels with readings to 1. Edge detection was then performed on the binary image to extract the shape features Fig. 3.7(b). The Hough Transform was then applied to the edge image and the peaks corresponding to lines that were ±5 degrees to the positive y-axis were chosen as they corresponded to the sides of the trunk Fig. 3.7(c). As small line segments were intermixed with larger ones due to the non-uniform shape of the trunk, only line segments that were larger than 100 pixels in length were chosen from the corresponding peaks to overcome this issue. These segments were then used to obtain the corner values for the bounding box that would indicate the location of the tree trunk Fig. 3.7(d). This detection was then combined with the results from the colour image detection to obtain the tree trunk locations. The steps of the depth image processing are shown in the flowchart in Fig. 3.8.

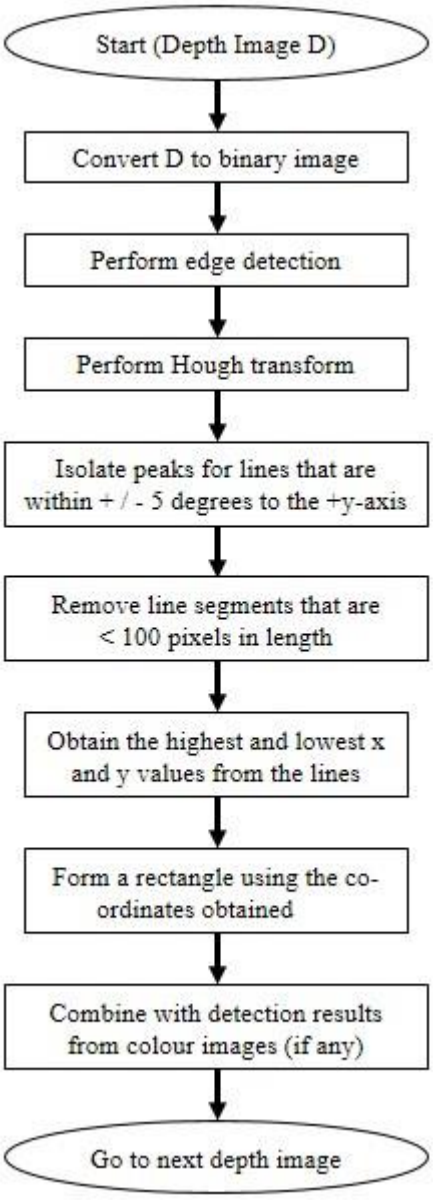


Fig. 3.8 - Flowchart of the depth image trunk extraction process

3.2.4 Nearest Tree Localization

Though the tree detection part of the OPTD algorithm worked well in the detection of the trees in the image, it was unable to pin-point the nearest tree during the day when the depth information was not available due to the presence of sunlight as the light interfered with the KINECT's depth sensor. The use of only colour image processing, led to the detection of multiple trees, and without the depth data, the created system could not narrow down the nearest tree. To obtain the location and distance to the nearest tree, an additional algorithm was created to be run after all the trees in the image had been detected. In addition to this, the distance to the tree as well as its orientation as compared to the robot was needed for the purpose of path planning.

To obtain the distance to the nearest tree, the first step was to narrow down the detections and identify the nearest tree before estimating its distance. Based on observations of the output of the tree detection algorithm, the nearest detected tree was determined to be the one with the largest bounding box in an image. In addition to deciding the nearest detected tree, the position of the centre of the tree with respect to the centre of the image was also recorded. This information was then used to estimate the angular position of the tree from the robot. In cases where the tree lied close to the sides of the image, the robot was slowly rotated on the spot until the tree was detected to be closer to the centre of the image. This was done so that a more accurate distance to the tree could be obtained the closer it was to the centre of the image. It also helped when inserting the target point in the path planning algorithm as the target point could then be considered to be straight ahead, instead of at an angle to the robot. This process was done by cross correlating the pixels of the initially detected tree with the pixels in new images obtained when the robot was rotating, so that the detected tree could be tracked during the rotation.

Since in any image, the size of the object varies depending on the distance of the object to the camera, in this case, the closer the tree was to the camera, the larger it would appear and vice versa, this principle could be applied to finding the distance to the tree. By using the relationship between the pixel width of the tree in the image and the actual distance between the camera and the tree, the distance to the detected oil palm tree

could be estimated by using only images. By obtaining the measurements of the distance between the camera and the tree and its corresponding width in the image, this relationship was found out to be linear in nature and was used to convert the width into the distance. The graph displaying the relationship between the pixel width of the tree in the image and the actual distance to the tree from the robot in meters is shown in Fig. 3.9 below.

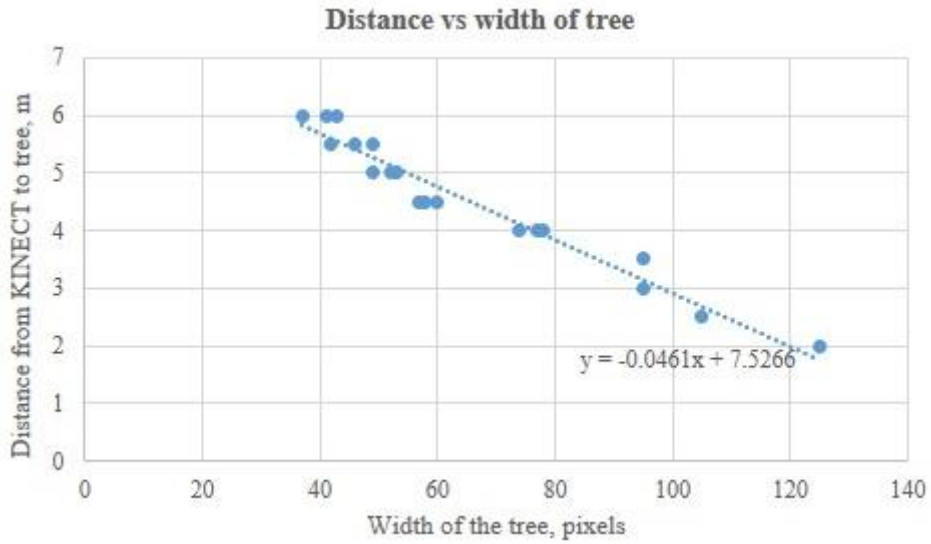


Fig. 3.9 - Graph of relationship between pixel width of trunk and actual distance to the trunk

The algorithm of the proposed image processing technique to obtain the distance to the nearest tree is explained below along with Fig. 3.10 and Fig. 3.11 to help explain the process.

1. Crop – After the colour image is processed by the tree detection algorithm, the centroid of the largest bounding box is used as a reference point to crop out the target tree (Fig. 3.11(a)). The cropping is done by extracting a rectangle with the height of the detection box and a pixel width of 60, centred by the reference point. This crop width value was chosen because the average pixel width of the experimented tree did not exceed 60 pixels.
2. HSV colour threshold – This threshold is used to eliminate the ground area in the cropped image. It is done by observing the Hue histogram of the HSV space and eliminates the best hue range value that describes the ground area (Fig. 3.11(b)).

3. Binarization – The image is converted to binary to simplify the RGB tree image (Fig. 3.11(c)). Local adaptive thresholding is used for the conversion by computing the local mean intensity in the neighbourhood of each pixel and using it as a threshold for binarization.
4. Fill holes – The converted binary image contains darker regions that do not convert accurately and are therefore cleared by filling in the surrounded regions to further filter the image (Fig. 3.11(d)).
5. Open morphology – In order to exclude the tree from background, a rectangle structuring element is used in open morphology to remove the background (Fig. 3.11(e)).
6. Closed morphology –This is used to smoothen the internal area of the tree so that the highlighted area looks like the structure of a tree (Fig. 3.11(f)).
7. Width extraction – Finally, the average width of the tree is extracted from the resulting isolated tree trunk image via the equation:

$$distance\ to\ tree = -0.0461 \times trunk\ width + 7.5266 \quad (15)$$



Fig. 3.10 - Output of all the detected trees

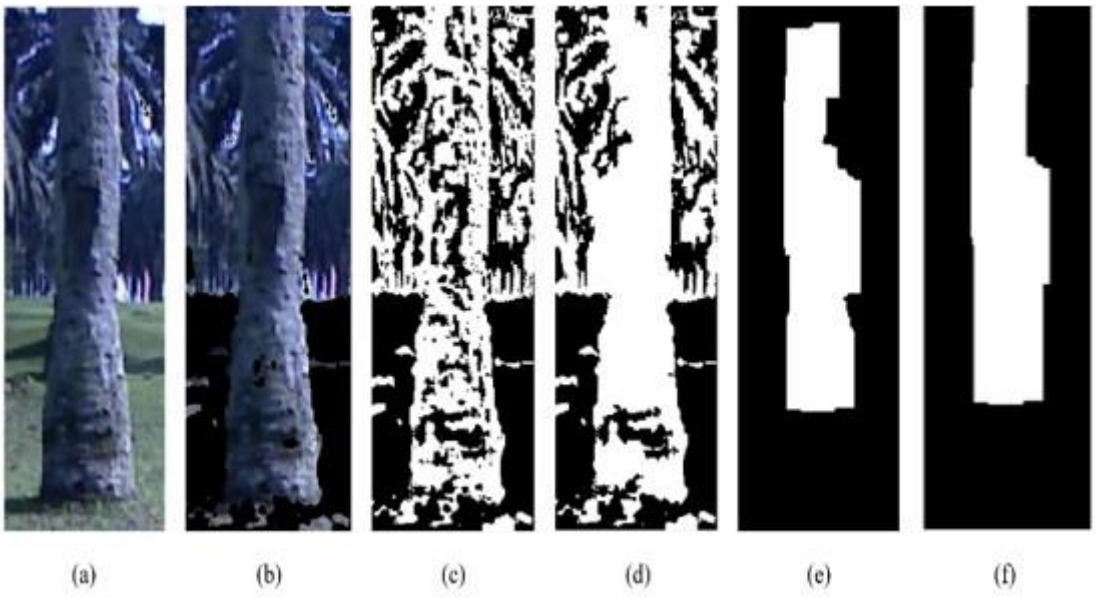


Fig. 3.11 - The process of extracting the approximate tree width for distance estimation

This information was then used in the path planning system to create a goal point, which is explained further in the next section.

3.3 Path Planning System

The moving target D* lite algorithm in this project was slightly modified from [9] to consider the kinematics of the robot rover. During path planning, the moving target D*lite algorithm planned a moving point instead of a robot rover and planned a collision-free path that did not consider how the robot moved or its size. In addition to this, in most cases the robot would not be able to follow the generated path with 100% accuracy as an actual object has mass, momentum, inertia and required a radius to make turns. Hence the algorithm had to be modified to take into account the kinematics of the robot by re-planning the path after each motion by taking into consideration the current position and orientation of the robot. If the robot had travelled further than a given threshold value from the path, the path would be recalculated from the new position. Due to the nature of the calculations performed by the algorithm, it was inconvenient to make the algorithm consider the robot rover's size in the computations made. To counter this, any detected object was inflated by adding on a value that was equal to the robot's width. This made the planned path give a sufficient gap for the robot to travel without colliding with obstacles. Although this modification to D* lite did not improve its computational performance, it was more robust in handling real time situations where the kinematics of the robot rover and trajectory controller were concerned.

3.3.1 Obstacle detection and mapping

The robot rover had 5 ultrasonic sensors placed at 5 strategic positions, as seen in Fig. 3.2 as the circles marked in red. The arrangement placed the ultrasonic sensors in the front and sides of the robot in a semi-circular shape to efficiently map any random obstacles within 4 meters away from the robot. For the mapping part of this system, the only inputs were the detected obstacles, the goal point / detected tree, the robot's current position and heading as well as the previously travelled path.

Since the projection of the ultrasonic sensors was a cone shape, all the ultrasonic sensors were installed at an appropriate height from the ground to prevent any interruption of measurements caused due to the reflection off of the ground. The sensors were

programmed to take measurements in sequence to ensure that the wave of each sensor did not interfere with another. As the ultrasonic sensor projected a cone shaped range and the area it covered increased with distance, the exact size of the object was not obtained from a reading. To ensure a clear path was available, the maximum size of the object was considered by inflating the object to the width of the cone at the detected distance. Moreover, as the path planning algorithm did not consider the size of the robot when planning the path, this object inflation also added on a value that was equal to the width of the robot, to ensure that the planned path circumnavigated the object successfully.

Finally, to avoid the possibility of a random erroneous reading caused due to a non-obstacle appearing as one or in the case of a passing object, a simple confirming method was used in mapping the obstacles detected by the ultrasonic sensors. This method used two states to confirm if the detected obstacles were confirmed as one. Any obstacle detected for the first time was put in the map with a pending state which turned into a confirmed state only if it was detected twice in the same location. This also helped prevent certain readings where the angled ground surfaces sometimes appeared as obstacles, but were later removed as they were not detected a second time after motion.

3.3.2 Integrated path planning system

All the previously explained subsystems were combined to form an integrated path planning system that was capable of planning a clear path through an oil palm plantation. The integrated system began by capturing 5 frames from the video stream and passing them to the tree detection algorithm. The tree detection algorithm analysed and identified the oil palm trees. If there were no oil palm trees detected in the 5 consecutive images, an imaginary target point was set a few meters in front of the robot, which was then moved around in search of oil palm trees. When trees were detected, the tree with the largest detection box was chosen, the robot centred to face it, and the distance to the tree calculated. All this was done by the OPTD algorithm mentioned earlier. After the location was found, the targeted tree's location was mapped into the graphical map and passed to the path planning algorithm. Before the optimal path was planned, the surroundings of the robot rover was checked for obstacles via the

ultrasonic sensors and marked on the map. The path planning algorithm then planned an optimal path to the target point which was then fed to the trajectory controller. The trajectory controller moved the robot rover to follow the optimal path. At each iteration, for every 2 seconds that the robot rover travelled, the new location of the robot was updated by using the rotary encoders and IMU to calculate the distance moved and the change in heading. 2 seconds was chosen as the time taken for a singular motion, as the distance the robot moved within 2 seconds was able to be tracked accurately by the sensors. If the motion had been for longer, more inaccuracies could have been added up via minor errors in readings made, and as localization in the outdoor plantation environment was difficult, the 2 second motion was set for the entire project. Once the distance moved was estimated and the new robot position updated, a new scan of the area was taken to search for new obstacles. This process of scanning for objects and updating the robot location was repeated until the target point was reached.

The exception to this procedure occurs if either the robot encountered an obstacle blocking the path or if the robot had moved off of the planned path. A certain degree of tolerance was coded in, where if the robot position was just slightly off the path but was within the tolerance range, the path need not be re-planned. In either of these cases, the path planning algorithm was used to re-plan the path by taking the new factors into consideration. The entire procedure is displayed in the flowchart in Fig. 3.12.

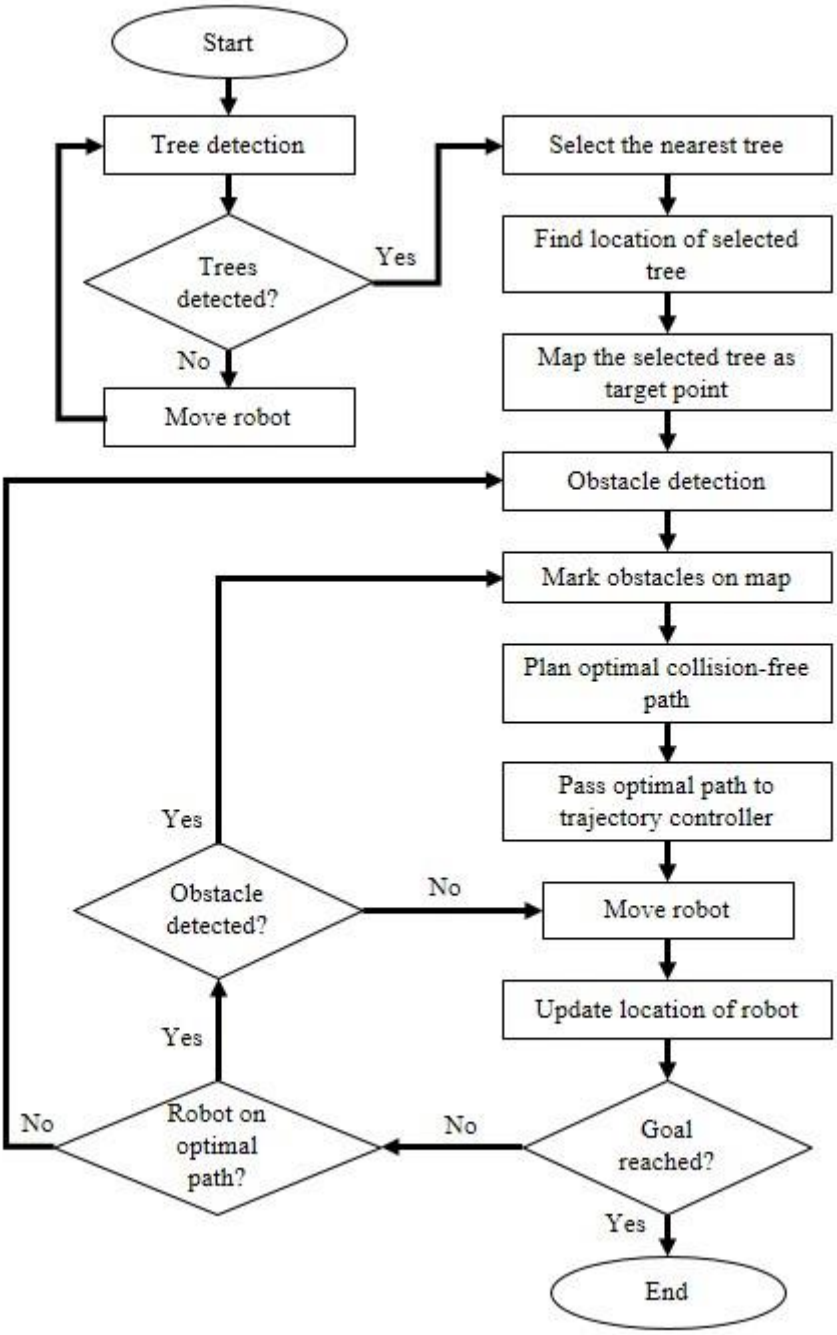


Fig. 3.12 - Process flow of the path planning system

3.4 Trajectory control system

This section is focused on controlling the 4 wheel skid steer mobile robot by using a trajectory control algorithm as described in Section 2.4. As mentioned earlier, due to the robot being a skid steer type and it having to be able to move in a variety of motions depending on the provided path, an adaptive algorithm that was capable of incrementally learning newer motions while retaining an accurate path following capability was required. As a solution, a novel adaptive Neural Network controller, capable of controlling the robot's motion was designed. The ESOINN was chosen due to its capability in organising unlabelled data and performing incremental learning which allowed it to retain existing knowledge it gained from previous trainings while learning new information. As a result, it could adapt and respond to new working environments. It was also able to eliminate the use of a full kinematic and dynamic model of the robot, as the network could create its own model during training via unsupervised learning.

The proposed method produced two improvements compared to the ESOINN, both stemming from the inability of the ESOINN to produce continuous outputs. The first improvement was to convert the discrete outputs to continuous outputs, so that outputs could be obtained from input data that was present in between learnt nodes via interpolation. Interpolation was achieved by the application of the Inverse Distance Weighting (IDW) method mentioned in Section 2.4. This was extremely important, as during the control of the robot, varied motions would be required based on the terrain and distance to navigate. If only discrete outputs were produced, the robot would only be able to move the learnt motions, instead of adapting to the required motion. The second improvement was related to the issue where the ESOINN was incapable of learning new data nodes when placed in a closed system loop without new external data. In this case, the ESOINN would only be capable of increasing the accuracy of its current nodes or removing nodes that had not been updated after a number of iterations of incremental learning, while being unable to produce new nodes. This was also directly solved by enabling continuous outputs to be produced, as this would essentially be creating new temporary nodes (virtual nodes) during the output stage, which could then be learnt as completely new nodes in the next iteration of incremental learning.

3.4.1 Training Data Collection

Training samples were collected to first train the neural network model. Although ESOINN did not require any labelled data, the robot had difficulty in navigating without any prior knowledge. Thus, training samples were collected and used to train the neural network model and give the robot a proper head-start. The collected samples were used for training both the ESOINN and IDW-ESOINN algorithms. The training samples collected were in four dimensions.

1. The PWM signal given to the left wheels
2. The PWM signal given to the right wheels
3. The direct start to end point distance moved as a result of the PWM combination
4. The radius of the curve the robot moved in

The wheels on one side of the robot were run on the same Pulse Width Modulation (PWM) signals. PWM signals were used to control the speed of the motor, via a motor driver by the on board microcontroller, an Arduino Mega. Different combinations of PWM signals to the driver circuit allowed the robot to move in different directions and speeds. These PWM signal inputs made up two of the dimensions of the training samples, which were for left side motors and right side motors. The path travelled by the robot could be accurately tracked by measuring its radius and distance travelled, making up the other two dimensions of the training samples. Each of the robot's motions were limited to only 2 seconds as explained in the previous section, as by keeping the time constant, all the other values (PWM, distance and radius) could be trained into the network without having to worry about a varied timescale.

Since PWM signals could be any integer values between 0 and 255 (0-127 rotated the wheel backward and 128-255 moved it forward), there was a total of 256 x 256 sets of possible left and right PWM combinations. Therefore, by having different PWM signal inputs to the motor driver of each side, the robot steered by skidding.

To obtain data for initial training, the distance and radius were manually measured after moving the robot for certain combinations of PWM signals. Measurement were taken only for 36 PWM combinations that moved the robot forward, backward and to the forward-right, by taking interspaced values within the range of PWM combinations. This was transposed to reflect the motion for when the robot moved to the forward-left, back-right and back-left, as ideally the robot would move the same distance and radius for the same difference in PWM signals, given that the load distribution was equal and the motors on each wheel were the same. This led to 136 sets of data (36x4 considering 4 quadrants of motion and then eliminating overlapping sets of PWMs for pure forward and backward motions). The PWM signal combination was sent to the motors and the robot was moved. Upon completion of the motion, the distance and radius of the motion was measured in centimetres by use of a measuring tape. A visual representation of the distance and radius with regards to the robot is shown in Fig. 3.13.

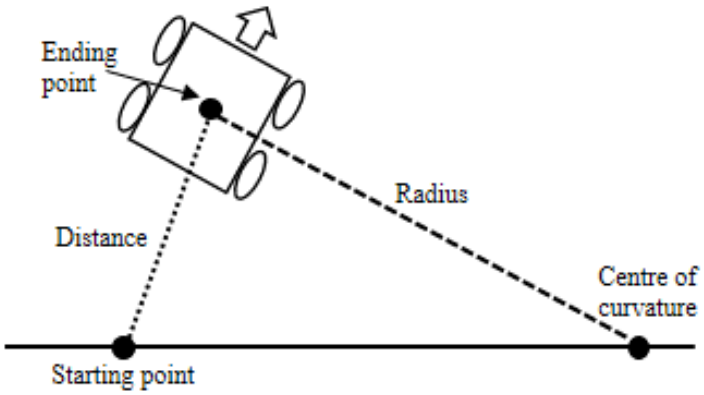


Fig. 3.13 - Measurement of Distance and Turning Radius

The distance is simply measured from the starting point to the centre of the robot once it had come to a complete stop. For the measurement of radius of curvature, a horizontal line perpendicular to the heading of the robot was drawn. This line passes through the centre point of the rover. Subsequently, after a single motion, another line which was perpendicular to the robot was drawn from the ending point. The point at which these two lines intersected was known as the centre of the curvature / turning.

Assuming the slippage of wheels was not significant, the distance from the centre of turning to either initial or final position of the centre of the rover should be the same, resembling the properties of a circle. This distance was known as the radius of curvature

/ turning. Several arbitrary conventions were established for ease of calculation and separating the data. Forward motions were taken as positive distance values and the negative distance values were backward motions. As for radii values, the right steer was denoted as a positive while a left steer was given as negative. An example of the collected data is shown in Table 3.3.

Table 3.3 - Sample of collected training data

Motion	Left PWM	Right PWM	Distance	Radius
Forward, Steer Right	255	200	60	420
Forward, Steer Left	200	255	60	-420
Backward, Steer Right	0	55	-60	420
Backward, Steer Left	55	0	-60	-420

3.4.2 ESOINN Initial Training

The 136 training samples with four dimensions were loaded into the algorithm for training. An output node of the algorithm represented a number of inputs that were related to each other. Similar inputs were grouped together in a node, because they were similar or nearer to each other in terms of their Euclidean distance. If more inputs were grouped together as a node, the resolution of the output node became smaller, thus precise control could not be achieved. So to gain more precise control of the robot, more number of nodes were needed.

The number of nodes were decided by 3 different parameters which were: age (age_{max}), lambda (λ), and denoise control (c). Age was the age_{max} parameter in step 8 of the ESOINN algorithm in Section 2.4.1 and lambda was the λ parameter in step 9. The denoise control parameter was assigned to remove those nodes that had only a single neighbour, where if that nodes had a value lower than that of (denoise control)*(mean value density of nodes), the node would be removed. In order to find out the combination that was best suited, each parameter was assigned a randomly generated number and the corresponding total number of nodes and accuracy of the nodes in their outputs were monitored. The training samples were trained with the parameters with the highest yield. The same parameters were also used during the incremental learning stage.

3.4.3 ESOINN Online Incremental Learning

When a robot is put into a real working environment, large amounts of data may be collected. It is impractical to load in the full set of training samples at each learning instance as this may take up a lot of computer resources and computation time. Therefore, it is more practical to only load new data for each learning instance, making the learning be incremental to previous learned data. The incremental learning for this project was done in batches, where one batch was the motion data for one successfully completed path.

For the incremental stage, all of the outputs from previous learning instances had to be loaded into the next learning instance. That is, not only the output node, but also the thresholds, connections and ages. This represented the previous learned neural network, which retained the ability of the network to eliminate noise. This enabled the input of only new data as training data during the incremental stage, as repeated learning of the previous data input could be avoided, thus maintaining a balance between plasticity and stability.

A point to note is that not only would incremental learning be useful in learning newer motions, but it would also be able to adapt to varying conditions over time that were influenced either by the terrain or the changes to the robot's kinematics or dynamics. The changes over time would be mirrored by the newer learnt nodes, which would enable the system to continuously perform while taking the current conditions into consideration.

For the collection of new data for this project, inputs were the distance and radius that the robot had travelled as well as the PWM signals that caused the motion. This data for an entire path was then concatenated to form new data for incremental learning. After a path was analysed and the robot finished its motion, the incremental learning function was called to update the network. The resulting network after learning was saved to replace the previous learned nodes. These nodes could then be used for the next path to be travelled.

3.4.4 The Novel IDW-ESOINN Algorithm

The drawback of the application of ESOINN in this project was that the output nodes were discrete in nature. The output of the network could only be values stored in the nodes. For instance, if an input to the network falls exactly on one of the stored nodes, the output was conveniently the value being stored in the particular node. On the other hand, since the nodes were discrete in nature, there was a high possibility that an input to the network did not fall onto any of the existing nodes and was located in between two or more nodes. In such cases, the output was taken directly from the nearest neighbouring node by finding the node with the shortest Euclidean distance to the data as this node had the highest similarity to the data.

An example of this can be seen in Fig. 3.14. Runge's function was chosen to test the estimation accuracy of an ESOINN trained using randomly sampled values of the function. Runge's function was used in this case as it is considered as a general test function in interpolation problems [133]. Runge's function is given by the equation:

$$y = \frac{1}{1+25x^2} \tag{16}$$

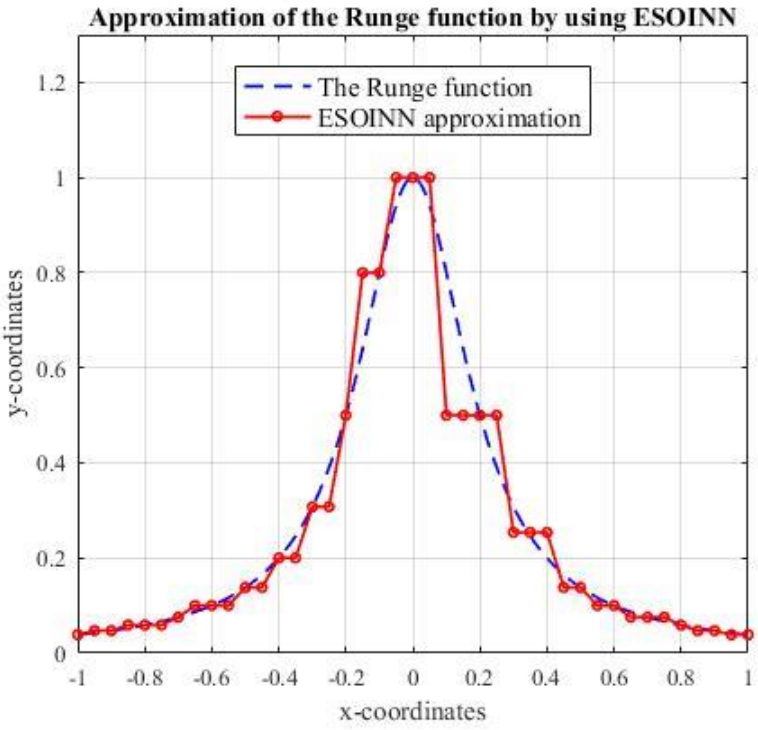


Fig. 3.14 - Approximation of Runge's function by using ESOINN

The solid line plot was the reconstruction of Runge's function based on data sampled at an interval of 0.05 in the x-axis of the original function. Due to the discrete nature and nearest neighbour classification feature of the ESOINN, a stepped function was produced instead of a continuous curve as seen in the figure, as only the learnt nodes were capable of being used as the outputs. This could be related to this project as each node described a particular motion that the robot could perform, and if no other motions could be made from the network output, the system would never be able to learn new motions, unless the user added in new measured values. This drawback was a fatal error for this application as this would limit the motions that the robot could move in. It would be impossible for the ESOINN to be accurate for every single case, since nodes were generated based on a number of distinctive training samples and for the control of the robot the driving PWMs could be any value from 0 to 255. If the algorithm was not improved or extended for continuous output, the only way to improve the performance was to collect 256^2 distinctive samples, which was extremely time consuming and would not be ideal for smart learning.

The existing algorithm of ESOINN was extended so that ESOINN possessed the ability to produce continuous output. The main idea was to estimate any output value that did not fall exactly on any of the nodes. This had 2 main effects, both of which were advantageous to the application. The first and obvious effect was that this would provide an output that did not exist in the current node set, which could basically be said that a virtual node had been created temporarily. The virtual node would essentially be a node containing the known and estimated values based on interpolation, formed at the time of extracting an output, at a region not containing an already existing node. This could also be considered as giving a continuous output. The second effect was that this virtual node could then be used in the next incremental training phase to update the current nodes for a better output, even leading to a completely new node. The data from the virtual node forms a perfect input for learning new data, as it contains a 4 parameter combination previously unknown to the network. This enabled the creation of more nodes in areas that were previously uninhabited, which would directly increase the possible motions that the robot could move to.

The extension that was proposed used the IDW method, where the values from an assigned number of nearest nodes to the input data were used, by averaging the combined values after they were weighted depending on the node distance to the input data point. For the use in this project, 2 new parameters were assigned when using the IDW method, which was *k* and *r*. *k* was the number of known nodes which the interpolation was to consider, as it was decided that having to consider all known nodes during interpolation would not give accurate results as the node spread was quite large. *r* was assigned as the search radius within which the *k* nodes should lie on, as if the known node lied at a large distance away from the unknown point, it would not positively contribute to the needed motion of the robot. The following steps describes the proposed algorithm, dubbed as IDW-ESOINN, used in this project.

1. The training data, x are fed into the neural network, ESOINN to obtain the output nodes. If the input data of the ESOINN has *n* dimensions, say $x \in R^n$, each node also consists of *n* dimensions of elements.
2. After the training of the ESOINN, a test input is fed into the ESOINN for output. The test input is at least one dimension less compared to the nodes because the missing dimensions refer to missing data which are stored in the nodes of the ESOINN. Say the query data (test input), y consist of *m* dimensions (*m* known elements) where $m < n$ and (*n*-*m*) elements of y are the unknown values to be determined using values stored in nodes. Compare each of the elements in each set of samples in y to the *m* elements in each of the output nodes. The comparison is done in Euclidean distance *d* using *m*-th dimension Pythagoras's Theorem,

$$d = |y_j - x_t| = \sqrt{\left(y_{1_j} - x_{1_t}\right)^2 + \left(y_{2_j} - x_{2_t}\right)^2 + \cdots + \left(y_{m_j} - x_{m_t}\right)^2} \quad (17)$$

   where j and t are the subscripts referring to the point of interest and node being studied respectively.
3. If the Euclidean distance *d* of *m* elements of a point *j* in y space to *m* elements of a node (say node *t*) is zero, go to step 4, else go to step 5.
4. The missing (*n*-*m*) elements of the point of interest in y space is exactly the same as the (m+1th to n^{th}) elements of the node *t*, making the output for this test input the values in node *t*. Go to step 2 and proceed with other points in y space.

5. Since the Euclidean distance is greater than 0, the point of interest does not fall on any of the nodes. Assign number of neighbours, *k*, a search radius, *r* and power parameter, *p*, for the IDW calculation. Determine the Euclidean distance *d* of the point of interest to each of the nodes based on *m* elements.
6. Take the first *k* lowest distance nodes as the neighbours of the point. If any of the neighbour nodes has a distance greater than the search radius, remove the node as the neighbour. Update the number of neighbouring points, *k*.
7. Divide each of the (m+1th to n^{th}) elements in neighbouring nodes $i = 1$-k (which are the missing data in the input test) by respective Euclidean distance d_i of respective node to the point of interest, point j. Find the IDW sum of ratio by:

$$Sum\ of\ ratio\ (SR) = \sum_{i=1}^{k} \left(\frac{z_i}{d_i}\right) \quad (18)$$

where z is any of the ((m+1)th to nth) elements of neighbouring nodes which is considered at the moment

8. Raise the power of each Euclidean distance of respective neighbouring node to the point of interest by *p*. The IDW reciprocal of powered distances is summed up by:

$$Reciprocal\ of\ powered\ distance\ (RP) = \sum_{i=1}^{k} \left(\frac{1}{d_i{}^{p}}\right) \quad (19)$$

9. The unknown element could be found using the IDW relation as follows:

$$Value\ of\ unknown\ element = \frac{SR}{RP} = \frac{\sum_{i=1}^{k}\left(\frac{z_i}{d_i}\right)}{\sum_{i=1}^{k}\left(\frac{1}{d_i{}^{p}}\right)} \quad (20)$$

10. If $(n - m) > 1$, there are more than one unknown element in each set of samples in y space, repeat steps 7 – 9 by increasing z to the next dimension of query. If $(n - m) = 1$, go to step 11.
11. After all the elements of a sample is known, this forms a new temporary virtual node. This is then inserted into the network via incremental learning (step 2-11 of the ESOINN algorithm mentioned in Section 2.4.1). If there are samples remaining with unknown elements, proceed with other points in y space until all unknown elements of every point in *y* space are determined.

An example of a situation requiring the IDW calculation is shown in Fig. 3.15 and the relevant calculation to find the value of the unknown node is shown in the equation that

follows. A flowchart of the proposed IDW-ESOINN algorithm is shown in Fig. 3.16 in the next page.

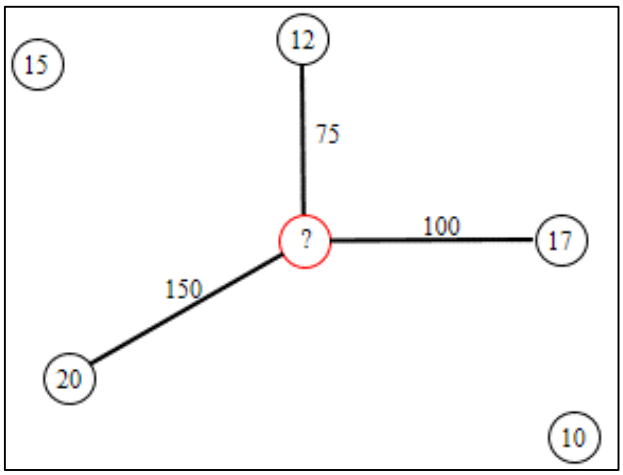


Fig. 3.15 - Example of estimation of unknown values based on values from 3 Neighbours

$$Z_{query} = \frac{\sum_{i=1}^{k}\left(\frac{z_i}{d_i}\right)}{\sum_{i=1}^{k}\left(\frac{1}{d_i{}^{p}}\right)} = \left(\frac{\frac{12}{75}+\frac{17}{100}+\frac{10}{150}}{\frac{1}{75}+\frac{1}{100}+\frac{1}{150}}\right) = 13.2\ (given\ p = 1, k = 3, r < 1000)$$

For the IDW-ESOINN algorithm used in this project, the IDW used had the following parameter values: p = 1, k = 3 and r = 1000. These values were found to give the best interpolated values as outputs after running tests to find the optimum parameter values.

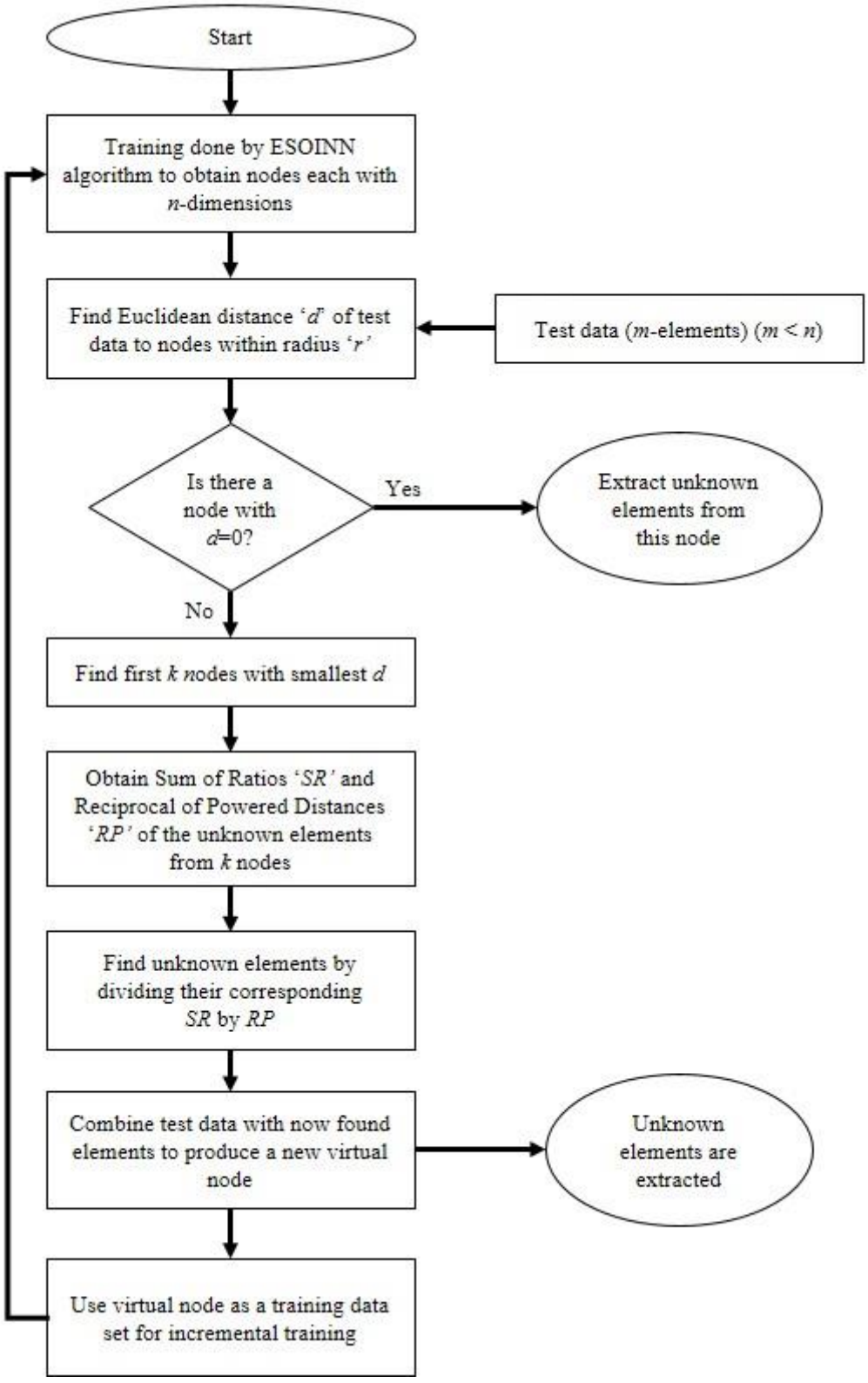


Fig. 3.16 - Flowchart of the IDW-ESOINN algorithm

3.4.5 PWM output extraction

During motion, the robot's distance to a point in the path as well as the radius of motion to reach that point needed to be calculated so that the corresponding PWM signals for that motion could be obtained from the trained network. The PWM signals could then be sent to the robot's motors to make the robot perform the required motion.

(A) Distance Calculation

A path is formed with many points along the path. For any two points (x_1, y_1) and(x_2, y_2), their distance was calculated with the formula below, which represented the simple Euclidean distance between them.

$$Distance\ (D_{path}) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (21)$$

(B) Radius Calculation

In common cases where slippage is not a serious issue, the turning of the rover should form an arc with a radius. Thus, by using the properties of circle, the radius of the turn could be found easily. A circle is defined by a centre and a radius. Radius is obtained by taking the distance between every point on the circle to its centre. Therefore, three points are needed to define a circle or an arc successfully. The calculation of the radius was based on the fact that the start point, end point and centre of rotation formed an isosceles triangle, with the equal sides being the radius of the circle. A visual representation of this is shown in Fig. 3.17.

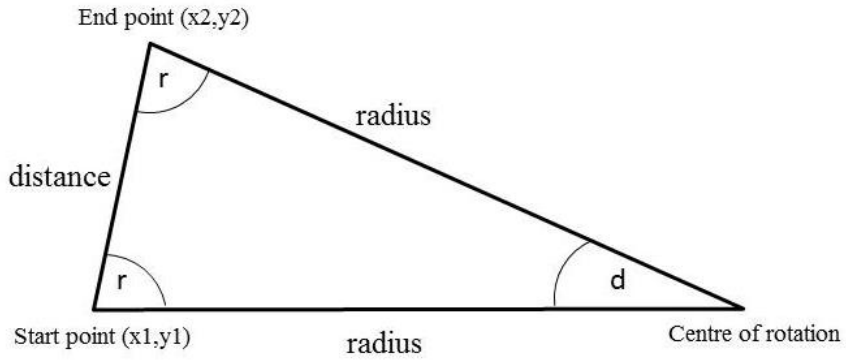


Fig. 3.17 - Angles used for the calculation of the radius

Using the properties of the angles of a triangle as well as the sine rule, the radius for the rotation were computed as follows.

$$r = \arctan(\frac{y2-y1}{x2-x1}) \quad (22)$$

$$d = 180 - 2 \times r \quad (23)$$

$$\frac{distance}{Sin(d)} = \frac{radius}{Sin(r)} \quad (24)$$

$$radius\ (R_{in}) = \frac{distance \times Sin(r)}{Sin(d)} \quad (25)$$

(C) PWM Calculation

Given a path composed of *N* points, named P_1, P_2,….., P_N where P is the Cartesian co-ordinate of the point, the motion of the robot along the path was calculated with the following steps:

1. Initialise the start and end points of the path as $P_{start} = P_1$ and $P_{end} = P_N$.
2. Calculate distance D_{path} of the path segment between P_{start} & P_{end}. If $D_{path} \leq D_{max}$, go to step 4, where D_{max} is the maximum distance the network has been trained to move the robot to.
3. Decrease the size of the path by 1 point ($P_{end} = P_{end-1}$) and repeat step 2.
4. Calculate radius R_{in} to move the robot by distance D_{path}.
5. Insert D_{path} & R_{in} into IDW-ESOINN as inputs and find the three nodes with the smallest Euclidean distance. (If there is a node that matches D_{path} & R_{in} exactly, then assign this as $Node_{out}$, and skip to step 7)
6. Use the IDW-ESOINN method to interpolate within the three nodes to create a virtual node ($Node_{out}$) to match D_{path} & R_{in}.
7. Extract the outputs: $D_{node}, R_{node}, PWM_{node.left}$ & $PWM_{node.right}$ from $Node_{out}$.
8. Send PWM signals to motors for motion control.
9. Save $D_{node}, R_{node}, PWM_{node.left}$ & $PWM_{node.right}$ to matrix Tr for incremental training.

10. Move onto next path segment by updating points $P_{start} = P_{end}$ and $P_{end} = P_N$ and repeating steps 2-10 till the entire path has been processed.

A visualisation of the calculated distance D_{path} and radius R_{in} can be seen Fig. 3.18.

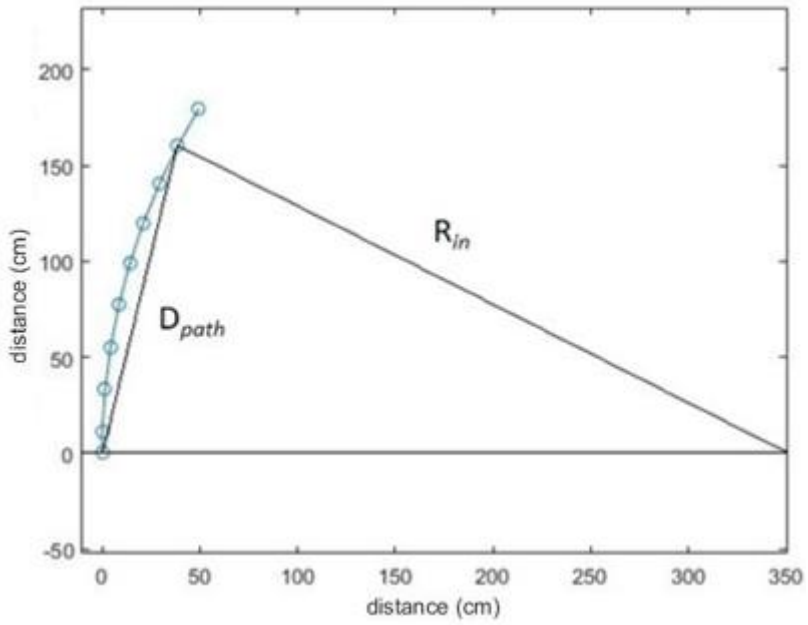


Fig. 3.18 - Visualisation of Distance and Radius

CHAPTER 4: RESULTS AND DISCUSSION

This chapter contains the results showing the performance of the designed systems, by testing the various aspects of their outputs in terms of accuracy, computation time or efficiency, in order to measure their performance. For the tree detector as well as the trajectory controller, comparisons have been made between the novel methods used and the original method, as well as other previously established methods in those fields. The main aim of this chapter is to prove that the created systems perform to satisfaction, while fulfilling their designed purpose. This chapter is split into 3 sub sections, with each focusing on a major subsystem of the navigation system.

4.1 Tree Trunk Detection System

The tree trunk detection system (OPTD) was first trained then tested to ensure that it was able to detect tree trunks with a high accuracy so that the system could be used without fear of missing out on any key detections. Testing was carried out on both static images and while the robot was in motion, as well as comparing the OPTD method to previously established object detection techniques.

4.1.1 OPTD System Training

Since the colour image processing segment of the OPTD method made use of the Viola and Jones detector, it had to be trained with images of oil palm tree trunks to identify them. 508 regions of interest (ROI) were selected from images taken from an oil palm plantation located near the Nottingham campus. The ROI consisted of tree trunks that were meant to be positive images for training. An example of selected ROI are seen in Fig. 4.1.



Fig. 4.1 - Positive regions of interest

As seen in the image, the nearest tree trunks visible in the image were chosen as ROIs. Negative ROIs were chosen from images with backgrounds resembling the background of the plantation environment. Negative ROIs had to have a similarity to the positive ROIs to ensure better detection accuracy as this helped the detector from wrongly classifying background items as the object to be detected. An example of a few negative ROIs can be seen in Fig. 4.2.



Fig. 4.2 - Negative regions of interest

For the initial stage, negative ROIs were selected randomly from the selection of negative images that were chosen for this purpose. For every subsequent stage, negative

ROIs were chosen by running the detector being trained on the negative images and selecting ROIs that were falsely identified as positive by the detector. The algorithm was set to follow this method to ensure the best background rejection rate.

For this method, the training was conducted with the following parameters values to first find the optimum detector to be used:

- ROI size (input size) of 119*32 pixels
- False alarm rate of 0.1,0.2,0.3,0.4 and 0.5
- Detection rate of 0.985,0.99 and 0.995
- Cascade stages of 5,8,10 and 12

The ROI size was limited to the mentioned size to limit the computations performed. Due to the nature of the feature search method as explained in the previous chapter where Haar-like features are resized and repositioned across the input in search of the best feature, the size of the picture sets the total number of features to be searched. The other parameters were set in various combinations to find the most appropriate setting.

During testing, the Viola and Jones algorithm searched the image by selecting sub-windows that were fed into the system. The sub-windows were scanned across each image, starting from the smallest possible size to the largest by varying the dimensions of the window. This method is termed as the sliding window method that is used as a basic search algorithm in most applications and is inbuilt in the Viola and Jones algorithm in Matlab.

A total of 60 detectors were trained by using a mixture of the parameters mentioned above and were tested to find out which ones had the best detection rates. After training and obtaining the results, a pattern was found that gave high detection rates when trained with the following parameters:

- False alarm rate: 0.1; Detection rate: 0.995; Stages:10
- False alarm rate: 0.5; Detection rate: 0.995; Stages:12

- False alarm rate: 0.5; Detection rate: 0.985; Stages:12

More detectors were then trained with the 3 parameter combinations in mind so as to investigate if detection rates improved if the false alarm rates were incremented / decremented in small amounts. It was found that the detector with 10 stages with a detection rate of 0.995 and false alarm rate of 0.145 gave the best performance. This detector was then used as the cascade detector for the OPTD system.

4.1.2 Detecting Tree Trunks on Static Images

Testing was carried out on 28 images of size 640*480 taken from the plantation with varying lighting conditions with a total of 65 trees to be detected from the images. Outputs from the original Viola and Jones detector and the novel proposed method (OPTD) were compared to outputs obtained by detectors using a Feed-forward Neural Network (FNN), CNN and SVM. Each detector was trained by using different input parameters so that an optimised network could be produced that gave the best detection rates for that method. The FNN used was trained using the Levenberg-Marquardt back-propagation algorithm with 3808 input neurons, 25 hidden neurons and 1 output neuron. The input size was decided by previously assigned dimensions of the tree trunk which was 119*32 pixels, which was 3808 pixels in total. The CNN was a 3 layer system with a filter size of 3*3 pixels per layer, and the 3 layers had a filter number of 4, 8 and 16 respectively. The SVM used was a binary support vector machine classifier using a polynomial kernel with an order of 2. All detectors were trained with 508 input images of tree trunks obtained from the plantation and 508 negative images for the FNN, CNN and SVM. The results obtained are shown in Table 4.1.

Table 4.1 - Results of various detectors on testing images

Detector	Detection Rate	False Positive rate
SVM [134]	44.0	27.0
Neural Network [135]	59.2	19.7
CNN [53]	56.6	46.0
Viola and Jones [8]	92.2	20.6
OPTD method	91.7	19.3

The detection rate was calculated as the number of trees that were successfully detected while the false positive rate was the number of incorrect detections made out of the total number of detections. As indicated by the highlighted results, the Viola and Jones detector performed extremely well compared to all the other detectors, although the false positive rate was slightly higher for the Viola and Jones detector compared to the Neural Network. When the pre-processing method from the OPTD system was applied

before the images were fed into the Viola and Jones detector, the results show that the detection rate decreases by 0.5% which is not ideal but the reduction of the false positive rate by 1.3% to have the lowest false positive rate justifies the usefulness of the technique. As mentioned earlier, having a low false positive rate helps save time that would otherwise be wasted by the robot having to perform more operations / motions before it could correct the mistaken detection.

The low performance of the CNN was a result of the low number of training data as well as the lack of clear features on the tree trunks, which made the detection via CNN not feasible for this project. The outputs from the Viola and Jones detector both with and without using the novel pre-processing method are shown in Fig. 4.3

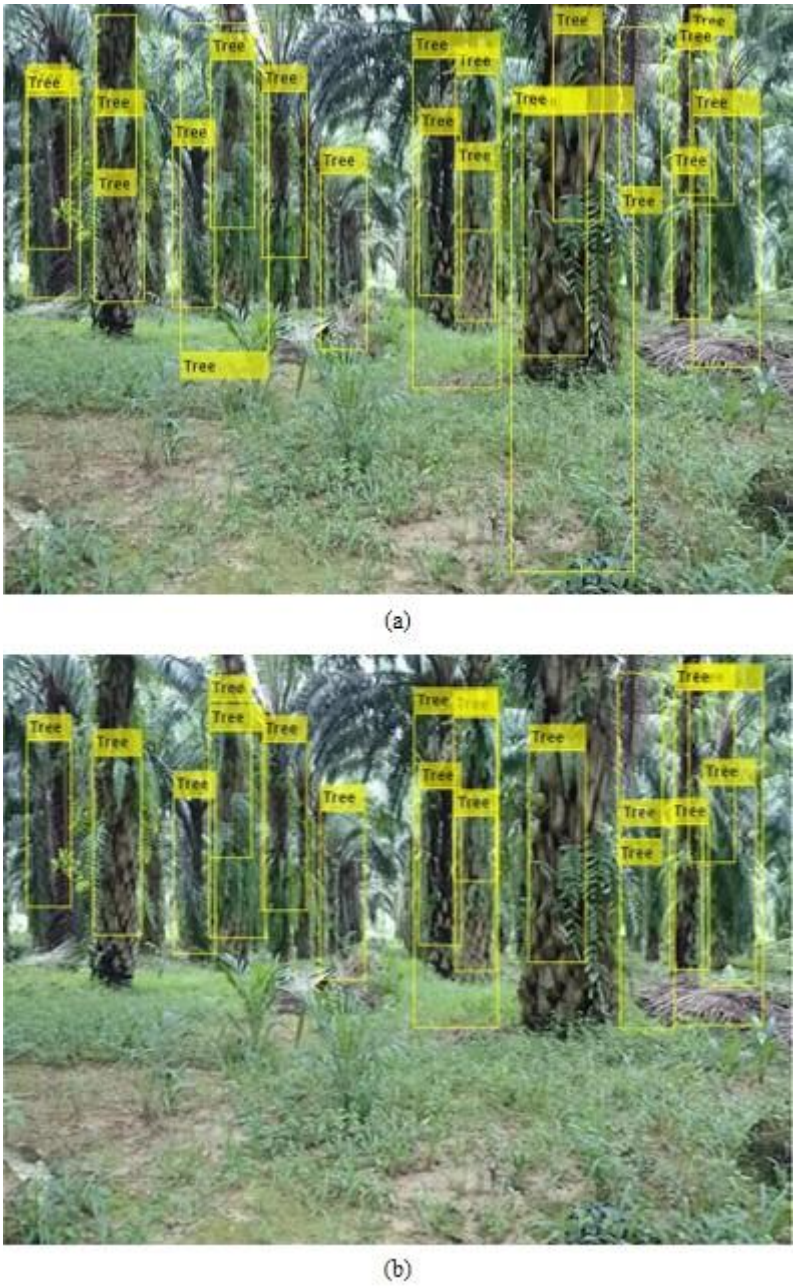


Fig. 4.3 - Static image test output: Viola and Jones detector output (a) without pre-processing (b) with pre-processing (OPTD)

It can be clearly seen that certain boxes that contain the ground that were mislabelled by the Viola and Jones detector as trees in Fig. 4.3(a) have been accurately judged as not trees when the pre-processing method was used in Fig. 4.3(b).

In addition to observing the detection rates, another test was done to compare the computation time taken to perform the detections, between the Viola and Jones detector with and without pre-processing. The results of this test is shown in Table 4.2.

Table 4.2 - Comparison of computation time per image between Viola and Jones and OPTD method

Detector	Computation Time (s)
Viola and Jones	0.0735
OPTD method	0.0706

The table shows the average computation time taken per image in seconds. The value was obtained by running the 28 static images through both methods and calculating the average time taken to obtain the detected trees from them. The results show that even though the OPTD method adds extra processes with the pre-processing, the time taken for the detections actually decreases in comparison to the Viola and Jones method. This would be a result of the smaller image size that is fed into the detector after pre-processing, leading to lesser number of sub-windows to be evaluated by the detector. The lower computation time is another point proving the benefit of using the OPTD system, as any time saved in computation is extremely useful while on the field.

4.1.3 Real Time Detection during Linear Motion

The robot was driven through separate plots of the plantation through the rows of trees as though it was moving from the beginning of a row to its end as well as approaching the trees placed at the ends of the rows when not travelling within a row. Images were extracted from the video feed at a rate of 3 frames per second and this input was used to test the detector on real time data. There was a total of 5 video feeds of varying lengths at different areas of the plantation from which the images were extracted. The results obtained by using the Viola and Jones detector with and without the proposed pre-processing method from OPTD, are shown in Table 4.3 and Fig. 4.4.

Table 4.3 - Results of running Viola and Jones detector and the OPTD method on real-time video feed

Detector	Detection Rate	False Positive rate
Viola and Jones	95.6	19.4
OPTD method	97.8	15.2

The results in Table 4.3 were obtained when the detector was run in an area where the trees were in a structured manner of rows as seen in Fig. 4.4. Due to the structured nature of the area, the contrast between the trees was clearer as a row-like arrangement could be observed. The accuracy of both detectors increased compared to the tests on the static images which were composed of various environments as shown in Table 4.1. In addition, the detection rate of the OPTD method as highlighted was 2.2% higher and the false positive rate 4.2% lower than the Viola and Jones detector, proving the better functionality of the proposed method. Considering the results from real time testing, it can be said that all results were highly influenced by the background tree density which depended on the area of the plantation as well as the orientation of the robot to the tree rows within the plantation.



Fig. 4.4 - Output from tests on video feed of structured environment (a) Viola Jones detector (b) OPTD method

4.1.4 Tree Detection Using Depth Data

Separate tests were conducted by using the depth data from the KINECT sensor as the inputs image had to be taken from close distances to the trunk to ensure that they were within range of the depth sensor. Recordings were taken from the KINECT sensor including both visual and depth information at different times of the day to test the reliability and usefulness of the depth information. All recordings were taken in the same area with the same approach pattern to enable accurate comparisons between data recorded at different times. The test data was extracted from those sets where the tree trunk was clearly visible in the depth image, generally at a distance of 2-3 meters from a trunk, resulting in a data set of 20 pairs of colour and depth images for each lighting condition. It was assumed that in the case where the tree trunk was not clear in the depth image, the robot would be moved closer to the object to get a clearer view so as to confirm if the visible object was a tree or not. All depth inputs were of the same size as the input images and data locations were adjusted to overlap the position of their corresponding pixels in the colour images. The results obtained are shown in Fig. 4.5

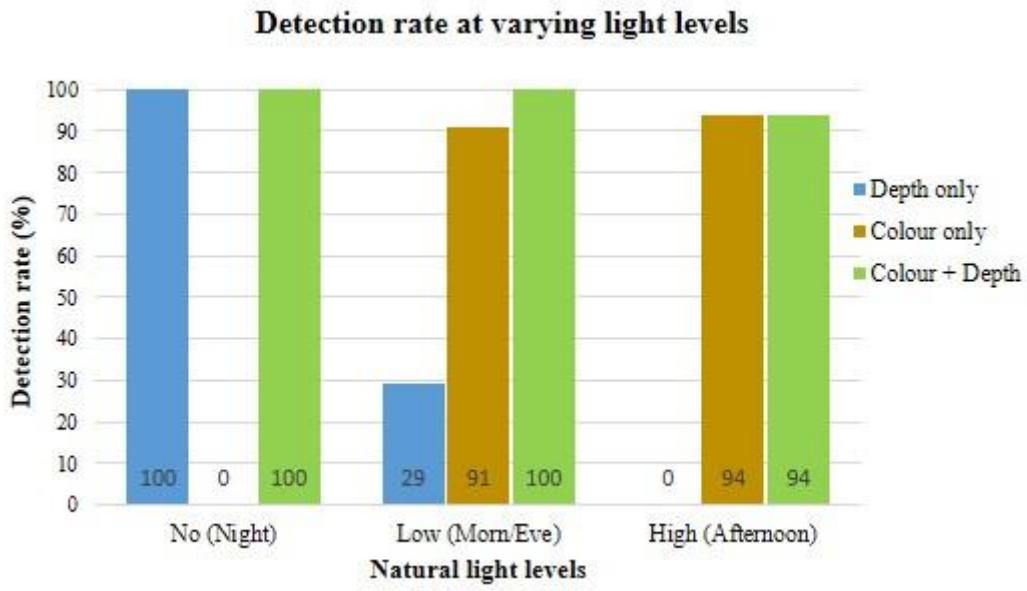


Fig. 4.5 - Detection rate of using the proposed OPTD method with and without depth data during various light levels

As shown in Fig. 4.5, the detections were split according to the level of light depending on the time of the day. Based on the typical Malaysian weather, the average value for

low light levels was below 10,000 lux and above 20,000 lux for high light levels [136]. The first set shows that at night when there was no light, there was no colour data input, resulting in 0% detections from colour images. In this case only the detections from the depth image processing were used, which had 100% accuracy as there was no interference to the infra-red light used to obtain the depth information. During the morning / evening as well as areas with shade, the low light produced enough colour data to make detections with an accuracy of 91% but the detections from depth data fell to 29% due to light interference. The combination of the detections made by each part gave a 100% detection rate as each system was able to detect trees that were missed by the other. Finally, when there was a high amount of natural light, generally around noon, no depth data was acquired due to the high level of interference with the depth sensor, resulting in complete reliance on colour image processing. With the higher level of light, the detection rate increased to 94%. This rise was due to the higher illumination of the trees and background, providing better features which enabled more accurate detections. This shows that the proposed OPTD method can work with a high accuracy throughout the day and the combination of the depth data to the colour image detections enabled a 100% confirmation on detected trees, which was generally difficult to achieve by only using images.

4.1.5 Real Time Tree Detection during Varied Motions

The accuracy of the detection of the OPTD system during various types of motion had to be investigated as well, as this impacted the images leading to a slight blur due to the motion. To do this, at different times of the day, mainly morning and evening, the robot was moved at a high speed, low speed or was rotated on the spot while the detections were taking place. A total of 35 trees were to be detected during forward motions and 30 trees for rotations, for each time of the day. The results of these tests are seen in Fig. 4.6.

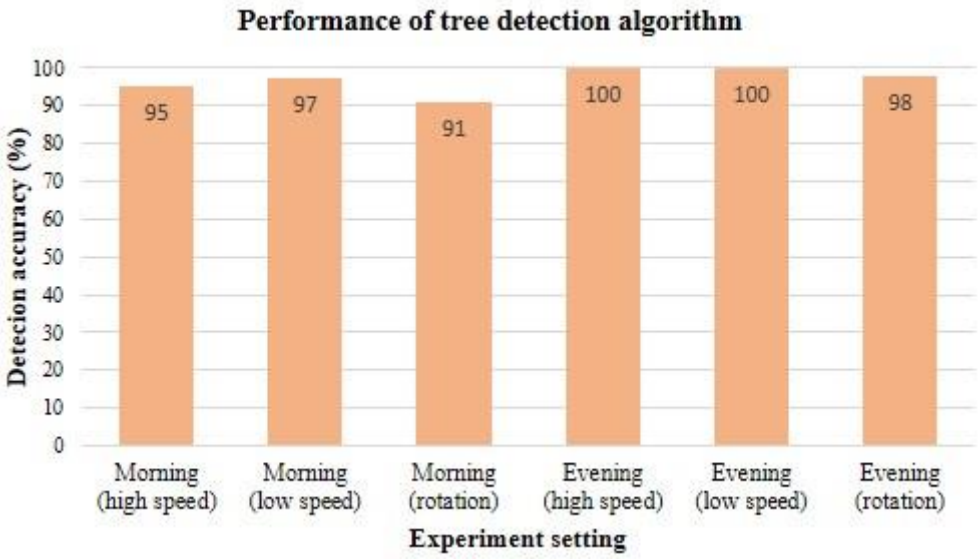


Fig. 4.6 - Real time performance of the OPTD algorithm during motion

The results show that the tree detection algorithm was able to perform with a very high accuracy even while moving, with a minimum accuracy of 91% regardless of the time of the day and the motion performed. The low accuracy during rotation as compared to the other motions was due to the fact that the image experienced a higher motion blur during rotational movement as compared to a regular straight movement.

4.1.6 Estimation of Distance to an Oil Palm Tree

The main objective of the process that was created to locate the nearest tree was to obtain an accurate estimation of the distance between the tree and the KINECT on the robot. Tests were run by placing the robot at different distances to different trees and running the nearest tree localization algorithm. The results in Fig. 4.7 show that the algorithm had good performance from 6 meters to 4 meters with an error rate of around 10-20%. Though the error did rise considerably when the robot was less than 4 meters from the tree, it was still within an acceptable range as within this distance the ultrasonic sensors would be able to detect the tree, leading to a confirmation on the distance.

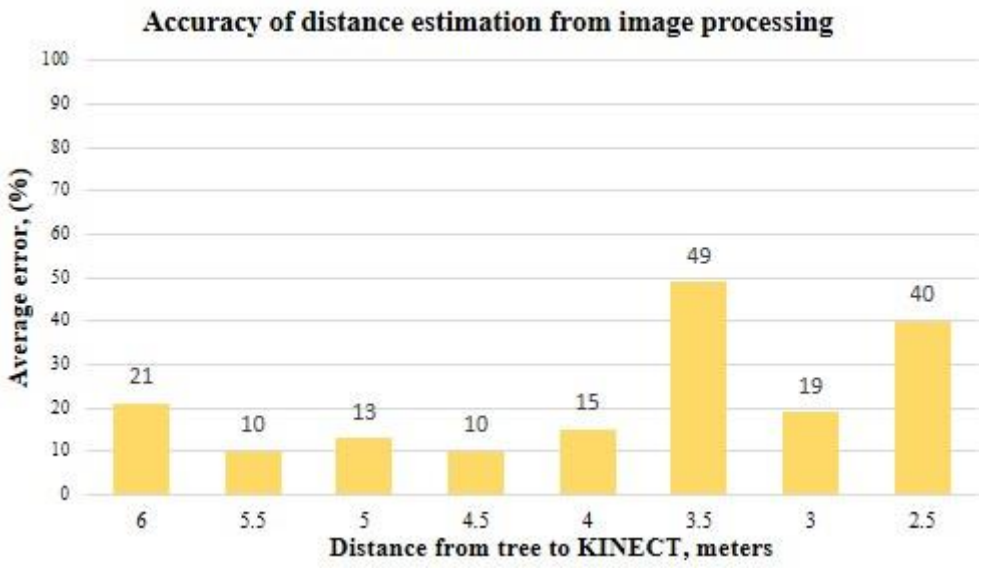


Fig. 4.7 - Accuracy of distance estimation from the tree to the KINECT from image processing

4.2 Path Planning System

The completed integrated path planning system was tested to ensure that it performed as expected. For the real time testing, the testing location was set up with obstacles placed in between the robot and the target position. The robot was positioned at a distance of 10 meters away from the target position and was allowed to autonomously navigate till it reached the goal point. Fig. 4.8 in the next page shows the result of the robot's attempt to reach the target point. The optimum path is shown as the purple line, the red 'x' marks any detected obstacles, the blue line displays the path travelled by the robot and the blue circles are the points at which the robot position was updated.

Fig. 4.8(A) shows the initial planned path. As the target was directly ahead of the robot, the path was a straight line. At the end of the second motion, an obstacle was detected, which was marked by the red 'x's. The obstacle appeared as a double line due to the object being inflated as mentioned in the previous section, to ensure that the planned path gave enough room for the robot to move around it. This caused the path to change so that the obstacle could be avoided, which can be seen in Fig. 4.8(B) as the purple line completely avoided the obstacles. It should be noted that the purple line originated from the point where the robot was previously at in Fig. 4.8(A), which shows that the path was re-planned till that point.

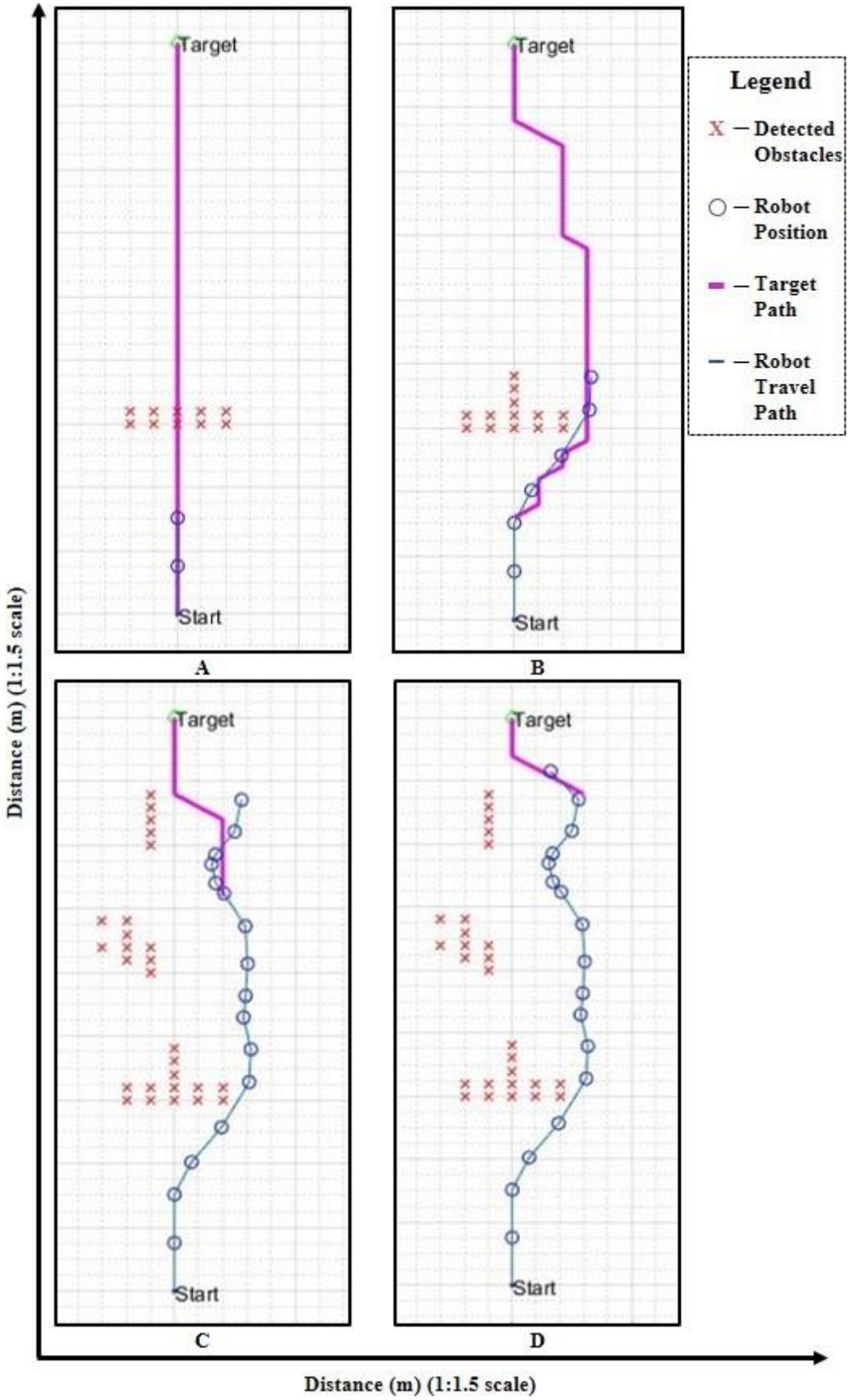


Fig. 4.8 - Results of running real time testing using the integrated path planning system

The robot managed to successfully avoid the obstacle as seen in Fig. 4.8(B) by the blue line and the extension of the red ‘x’s behind the original obstacle showed that the initially detected obstacle had depth that could not be seen from the front. This detection was made by the ultrasonic sensor placed on the left side of the robot. It can be seen that the robot did not follow the exact path planned by the algorithm, mainly due to the fact that the robot turned on a radius which was different to the angular turns produced by the D* lite algorithm. Even though there were minor deviations, the robot’s position remained on or very close to the planned path, causing the path to remain unchanged as there was no need for re-planning.

The robot closely followed the path shown in Fig. 4.8(B) for quite a distance (as seen in Fig. 4.8(C)), though at the later stages the robot deviated to either side of the planned path. This mainly occurred due to the robot making an error in one of the motions, which led to it trying to compensate and return to the original path. This is clearly displayed as the robot is shown to be moving in an oscillating manner so that the error may be reduced. In Fig. 4.8(D), it can be seen that the robot position had deviated too far off of the planned path displayed in Fig. 4.8(C), causing the path to be re-planned again to enable the robot to reach the target point.

By looking at the motion completed by the robot, it can be seen that the path planning works efficiently which led the robot on the most optimum path to the object. The obstacle detection system performed well in detecting the obstacle in front of the robot as well as the side of the obstacle once it had drawn parallel to the object. The issue of the robot overshooting the path on either side was mainly due to the error in the motion control of the robot by the trajectory controller as certain radii of turning were unable to be exactly mimicked. However, the path planning algorithm was still able to compensate and re-plan a suitable path for the robot to reach the target point, which shows that unplanned errors in motion in real time could still be handled by the designed system.

4.3 Trajectory Control System

For testing the full functionality of the designed trajectory control system, various aspects of it had to be investigated to test its feasibility in application. Test were carried out to check on the initial and incremental learning capability of the ESOINN as well as its performance in relation to other techniques, after which the proposed IDW-ESOINN method was tested to prove its comparative performance both in terms of learning and output accuracy against the original ESOINN.

4.3.1 Testing the Capability of the ESOINN in Trajectory Control

To settle on the ESOINN as the main method of control, investigations were first done to ensure that it could indeed be applied to trajectory control, while also narrowing down on the optimal parameters to be used to train the network so that the highest accuracy could be obtained.

(A) Obtaining the Optimal ESOINN Training Parameters

There were three parameters that could be adjusted in the ESOINN algorithm: maximum age (age_{max}), lambda (λ), and denoise control (c). These parameters were closely related to the end result of the algorithm. The full training set of 136 training samples as described in Section 3.4.1 were used to obtain the trained nodes. To evaluate the performance of the ESOINN in terms of node accuracy, the Root Mean Square Error (RMSE) was used to calculate the error between the set of training data and the corresponding output node of the ESOINN after training. The formula was as follows:

$$RMSE = \sqrt{\frac{\sum_{t=1}^{n}(\hat{T}_t - N_t)^2}{n}} \quad (26)$$

Where $\hat{T}_t = Values\ of\ training\ data$

$N_t = Values\ of\ output\ node$

$n = total\ number\ of\ samples$

The RMSE for each of the 4 values in the nodes were calculated, then an average was taken from them for a general RMSE value per node, as it was simpler to assess this

than to assess 4 different RMSE's per node per network. Tests were run to find the best combination of parameters, by using 200,000 combinations of varied parameter values, as each parameter had a range within which it could lie: age (1-100), lambda (1-100), denoise control (0.01-1). The preferred network would be one which would give a high number of nodes with a low RMSE. The high number of nodes was important so that the network would have a good resolution when it came to the number of motions it would be capable of accurately matching to. From the tests that were run, the 5 networks with the lowest RMSE were chosen for further investigation. The network parameters of those 5 networks are shown in Table 4.4.

Table 4.4 - Parameter values of trained ESOINNs with lowest RMSE

Network No.	Maximum Age	Lambda	Denoise Control	No. of Nodes	RMSE
1	2	73	0.5	64	11.155
2	3	73	0.5	63	11.489
3	2	85	0.5	61	11.409
4	2	70	0.5	62	11.765
5	1	69	0.5	61	11.966

Results from the table show that the parameters with the lowest RMSE had a low Maximum Age, with a high Lambda, while the denoise control parameter was the same for all 5 networks. Further tests were made after this to investigate how well these networks would perform after incremental training, as the results so far only indicated the performance after the initial stage of training. Incremental training was done by adding data from the ESOINN being run on 3 different paths. The paths had varied radii for their curved parts, to test the various motions capable by the robot and how well the controller could follow the given curves. After each path was simulated, the data from the ESOINN outputs was then added via incremental learning. The paths used are shown in Fig. 4.9.

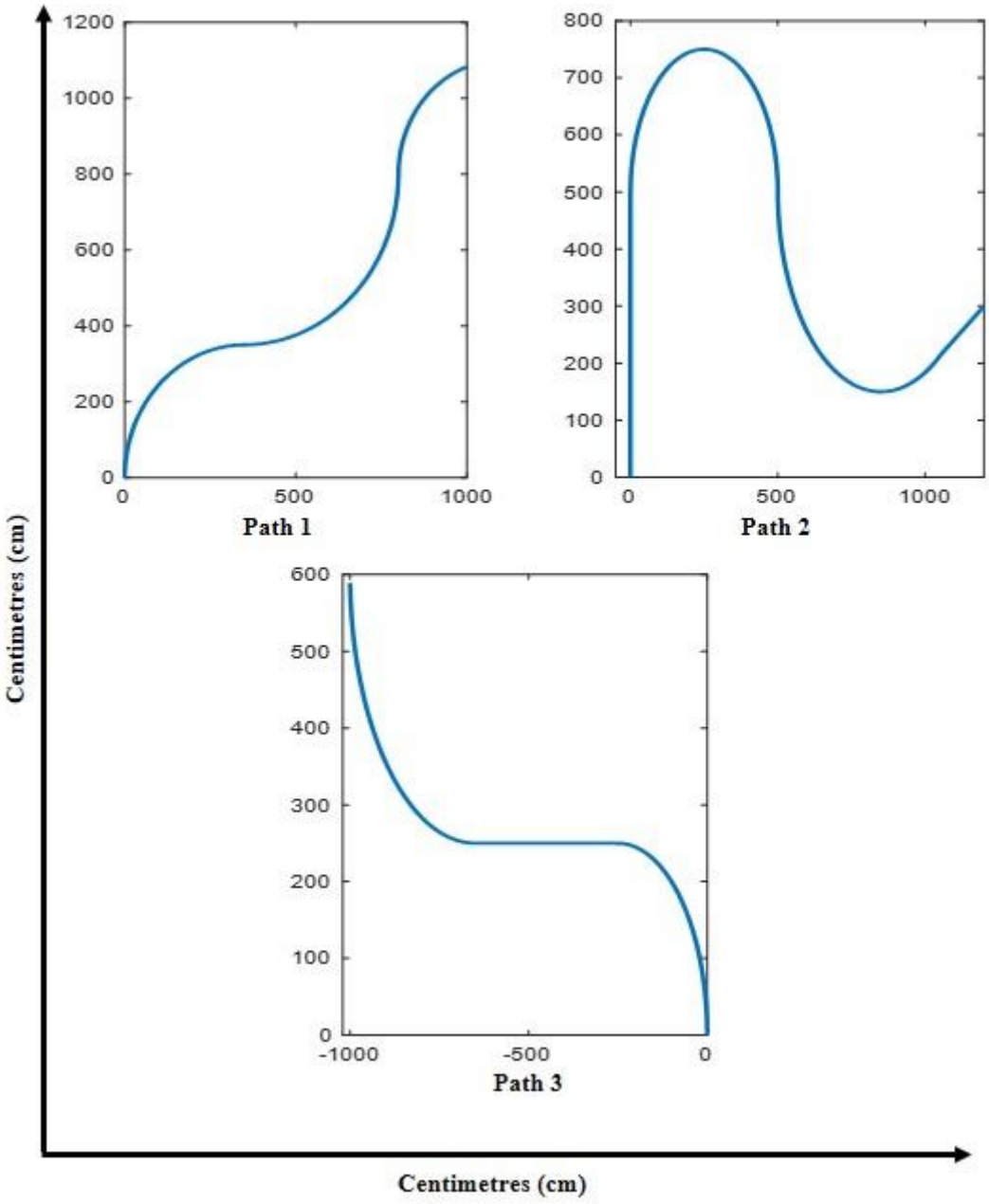


Fig. 4.9 - Constructed paths for simulated trajectory control

After the training was done, the number of nodes of the network as well as its RMSE with relation to the initial training data was compared again, to find the best network. The results from the incremental training is shown in Table 4.5 and Table 4.6.

Table 4.5 – Total number of nodes of each network after each stage of training

	Number of Nodes after training			
Network No.	**Fundamental / Initial Stage**	**1st Stage of Incremental Learning**	**2nd Stage of Incremental Learning**	**3rd Stage of Incremental Learning**
1	64	88	66	66
2	63	86	68	74
3	61	85	112	134
4	62	89	68	78
5	61	78	44	32

As seen in Table 4.5, network no. 3 had the best growth rate with relation to the number of nodes. This could be related to it having the highest value of Lambda as compared to the other networks. The high growth rate of nodes was extremely important, as this directly increased the number of motion options that were available for the robot to follow, which was crucial during the robot's operation.

Table 4.6 - RMSE of each network after each stage of training

	RMSE after training			
Network No.	**Fundamental / Initial Stage**	**1st Stage of Incremental Learning**	**2nd Stage of Incremental Learning**	**3rd Stage of Incremental Learning**
1	11.155	17.804	21.607	26.446
2	11.489	17.677	21.162	25.947
3	11.409	15.535	15.295	15.156
4	11.765	16.090	21.489	25.037
5	11.966	15.981	35.510	59.019

The results from Table 4.6 shows that network no. 3 had the lowest RMSE as compared to the other networks, and even showed signs of a decreasing RMSE after the 2nd stage of training, while it increased for all the other networks. The sudden increase during the 1st stage for this network can be explained as the sudden new addition of a varied path made a big shift in relation to the initial data, which made the RMSE increase. Looking

at the overall results, it was decided that network no. 3 had the best performance among all the trained ESOINN, and was chosen as the network for this project.

(B) Testing of ESOINN Outputs in Simulating Paths

A curved path was constructed as shown in Fig. 4.10 to test the performance of the chosen network with relation to its path following accuracy. The path was plotted with 25 points to form a curve with a constant radius, taking the plot origin as the original position of robot.

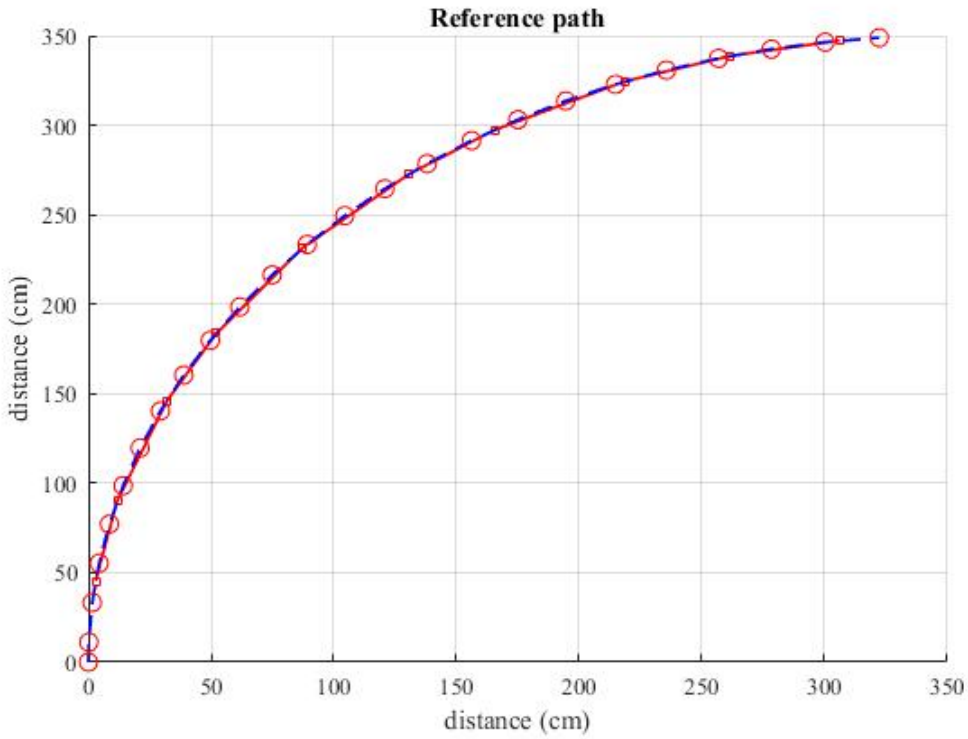


Fig. 4.10 - Reference Path

The network was used to simulate the path following motion, and the resulting path followed is shown in Fig. 4.11.

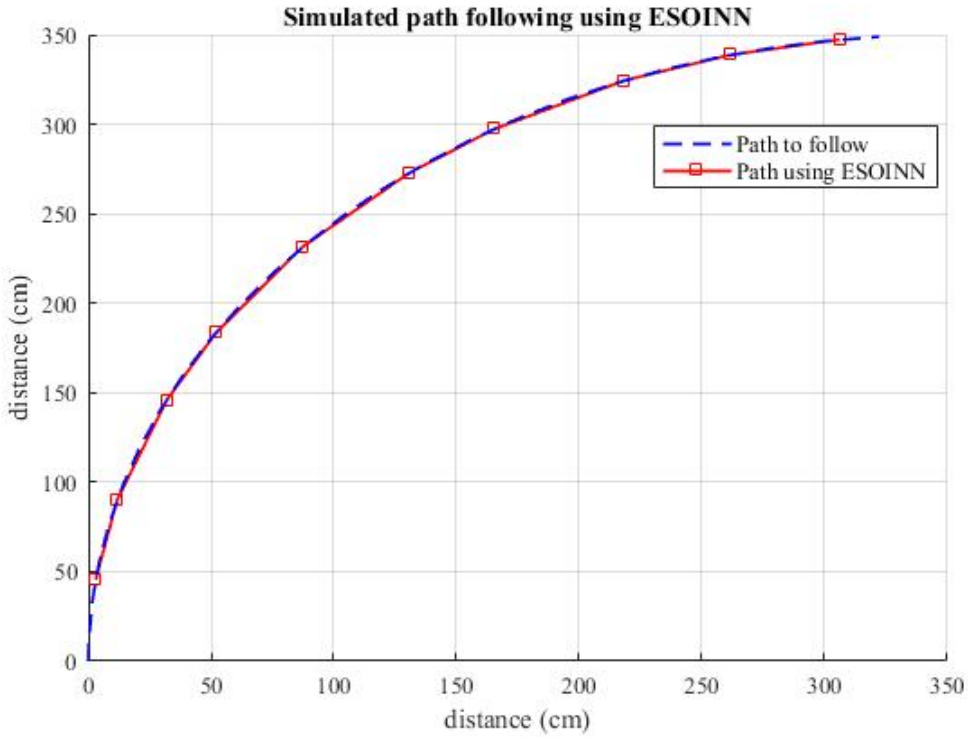


Fig. 4.11 - Comparison of Reference and Travel Paths using ESOINN

It can be seen that the path using the network was quite close to the actual path to follow, which shows that the trained network had good performance with relation to its outputs as well. The RMSE of the network while following this path, both after initial and incremental trainings were recorded and displayed in Table 4.7 below. The RMSE for the deviation in between the output and the path to follow was calculated as follows:

$$RMSE = \sqrt{\frac{\sum_{t=1}^{n}(\widehat{y_t} - y_t)^2}{n}} \qquad (27)$$

Where $\widehat{y_t} = travel\ point$

$y_t = original\ point$

$n = total\ number\ of\ points$

Table 4.7 - RMSE values during path following after incremental learning

Network type	RMSE (x-axis) (cm)	RMSE (y-axis) (cm)
After Fundamental / Initial Stage	6.618	7.456
After 1st Stage of Incremental Learning	6.658	5.588
After 2nd Stage of Incremental Learning	6.059	5.705
After 3rd Stage of Incremental Learning	3.474	4.036

As seen from the table, although the RMSE slightly increased for the x-axis after the 1st stage of incremental learning, and the y-axis for the 2nd stage of learning, the RMSE had a general trend of constant reduction. This shows that the network accuracy was maintained even after new data was added in, and even decreased significantly during other stages.

(C) Performance Comparison with other Methods

The performance of the ESOINN was then compared to other unsupervised learning algorithms: the Self-Organizing Map (SOM) method [126], the K-means clustering method [137] as well as a supervised learning feed-forward neural network [135] with adaptive weight learning (adaptive NN). Both SOM and K-means were similar to ESOINN, which used clustering to classify data, though they were unable to learn incrementally, thus for each incremental learning of new training samples, both algorithms needed to take in the whole set of training samples, which meant both the old and new samples. The adaptive NN on the hand was a supervised learning algorithm, but it was able to learn adaptively by updating the weights of the system with the addition of new data. The SOM had a map of 8x8 nodes and K-means had 64 clusters, as the idea was to keep the number of nodes / clusters similar to the ESOINN during its initial training where it had 61 nodes. The adaptive NN had 10 hidden layers, with 2 neurons each for input and output. First the training time was investigated for both initial training and subsequent incrementations when new data was presented. Table 4.8 shows the computation time taken for training between the networks. The lowest computation time for each stage of learning is highlighted to better showcase the results.

Table 4.8 - Comparison of computation times taken by the ESOINN, SOM, K-means and adaptive NN

Network type	Computation Time (seconds)			
	ESOINN	**SOM**	**K-Means**	**Adaptive NN**
Fundamental / Initial Stage	0.6061	0.4672	0.0423	0.3390
1st Stage of Incremental Learning	0.0467	0.5990	0.0251	0.1639
2nd Stage of Incremental Learning	0.0516	0.7171	0.0692	0.1645
3rd Stage of Incremental Learning	0.0071	0.8319	0.0294	0.1675

It can be seen that ESOINN took the longest time during its fundamental training, due to the larger amount of calculations to be made, but the subsequent trainings took a very short time as the network learnt incrementally as compared to SOM and K-means. SOM clearly took a longer time for each incremental stage, though K-means took a varied time as its clustering calculations were not that complex. The adaptive NN only performed better than the SOM with regards to training time overall, though it did take almost half the time during incremental learning as compared to the initial stage. Though when it came to the later stages of incremental learning, ESOINN took a much shorter time than all the other methods. ESOINN also had an advantage when it came to the nodes / clusters formed as compared to the other methods as it enabled the network to update the nodes and even remove outlying nodes without user input during each learning stage, unlike SOM and K-means where the number of clusters had to be specified prior to the training, which made it more complex when the data set was larger. Incremental learning wise, the ESOINN nodes had a faster training time to update their information as compared to the weight update procedure of the adaptive NN, which could be due to the adaptive NN having to do more calculations as it was supervised learning based. These networks were then run on the simulated path in Fig. 4.10 to compare their outputs. The results are shown in Table 4.9.

Table 4.9 - Comparison of RMSE for following a reference path

Network type	RMSE value (x-axis : y-axis) (cm)			
	ESOINN	**SOM**	**K-Means**	**Adaptive NN**
Fundamental / Initial Stage	6.618 : 7.456	6.938 : 6.009	6.437 : 5.056	3.351 : 4.574
After 1st Stage of Incremental Learning	6.658 : 5.588	5.577 : 8.421	4.140 : 5.007	7.379 : 4.108
After 2nd Stage of Incremental Learning	6.059 : 5.705	8.676 : 3.959	6.012 : 7.593	3.676 : 3.044
After 3rd Stage of Incremental Learning	3.474 : 4.036	8.236 : 6.117	5.209 : 5.078	4.356 : 4.238

As seen in the table, ESOINN had a consistent decrease in the RMSE for the path output by the network, while the other methods had a very inconsistent trend in their output errors. This meant that there was no guarantee that the other methods could give a better accuracy after incrementation and could in fact produce poorer results. Comparing the four methods, ESOINN was the only one that had a constant improvement after the addition of new information at each stage of training. This may be due to the fact that for SOM and K-mean, the full training and clustering had to be done again at every stage, making the methods create new clusters every time instead of updating already existing clusters, while for the adaptive NN, even though the overall performance seemed good, there was no guarantee that the update in the weights would prove to be a positive influence on future outputs, which made it unreliable. Reliability of system performance was an extremely important factor as during real-time operation the system's performance had to be able to improve and reduce the errors through learning.

All these tests proved that ESOINN was the best available choice as the trajectory controller for this application. It showed that the system could perform reasonably well, even without the need for a system model, which was generally necessary for other methods used in trajectory control.

4.3.2 Creation and Testing of IDW-ESOINN method

After the confirmation of the ESOINN algorithm in learning the trajectory data, the problem of its discrete outputs had to be addressed, and the solution as mentioned in Chapter 3 was to add the IDW interpolation method to make the outputs continuous, thereby creating a novel method which was the IDW-ESOINN. The tests conducted to evaluate its feasibility as well as its performance is described in this section.

(A) Approximation of Runge's Function

Before applying the IDW-ESOINN method to the trajectory control application, a simple test was carried out to test the performance of the approximation capability of using IDW with ESOINN to obtain continuous outputs as opposed to simply using the nearest neighbour values according to the original ESOINN. The function used for comparing this performance was the Runge's function [133], which was mentioned in Section 3.4.4. The Runge's function was used to create the dotted curve in blue in Fig. 4.12 in the interval of -1 to +1 in the x-axis. The data was sampled at intervals of 0.1 in the axis and this was used to train the ESOINN. Outputs were then extracted for intervals of 0.05 in the x-axis from the ESOINN, by either taking the values of the nearest neighbouring node (red-curve with circles), or by using the IDW-ESOINN method to interpolate data in between nodes (black curve with squares).

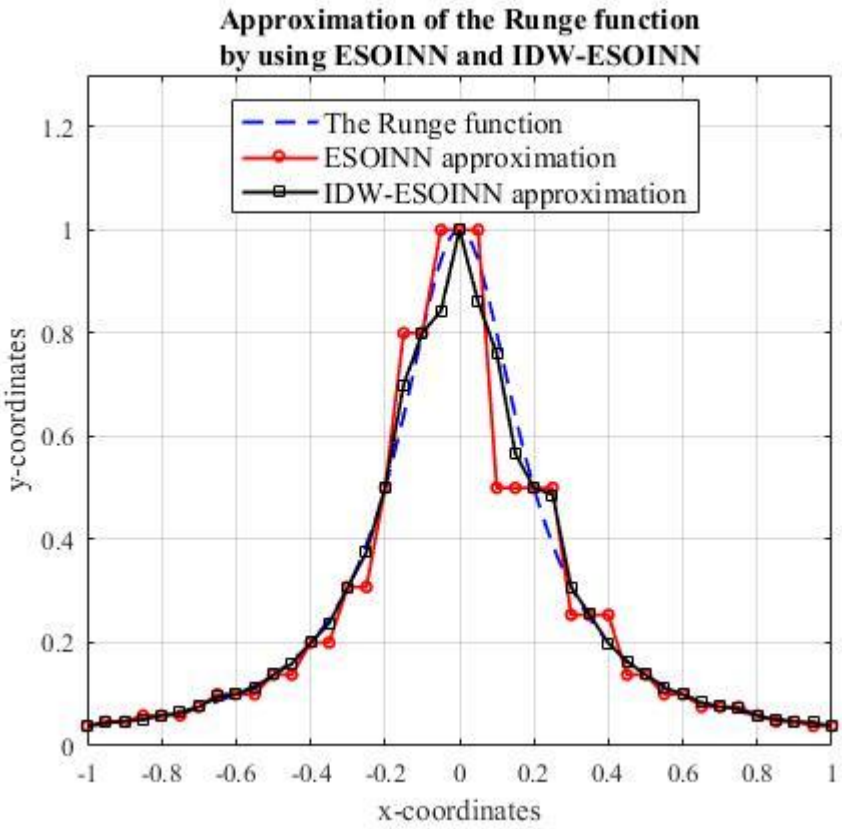


Fig. 4.12 - Approximation of Runge's Function by using outputs of the ESOINN and the IDW-ESOINN

The error between the Runge's function and the approximation was found by taking the RMSE between the actual point and the estimated point. Table 4.10 shows the RMSE values for the 2 approximations.

Table 4.10 - Comparison of estimation values of ESOINN and IDW-ESOINN

	Approximation using ESOINN	**Approximation using IDW-ESOINN**
RMSE	0.06468	0.02992

As shown in the table, by using the IDW-ESOINN method to obtain continuous outputs, the error caused by using the ESOINN method was reduced by 54%. This shows that the IDW-ESOINN method had the potential to greatly reduce errors caused by having discrete nodes when a continuous output was needed.

(B) Distribution of Learnt Nodes after Initial Training

As mentioned in section 3.4, the use of the interpolation method was not only to enable the ESOINN to provide continuous outputs, but to also create new virtual nodes in areas that nodes were not present in. This would lead to the creation of new permanent nodes after the next incremental training phase. To investigate this, the ESOINN trained in the previous section was used. The network used was the one after initial training without any incremental training. The initial node distribution of this network is shown in Fig. 4.13. Since it was impossible to visually display the nodes using all 4 dimensions of the node, only the distance and radius values of the nodes were used to display their distribution and it was limited to only those nodes with a positive distance value.

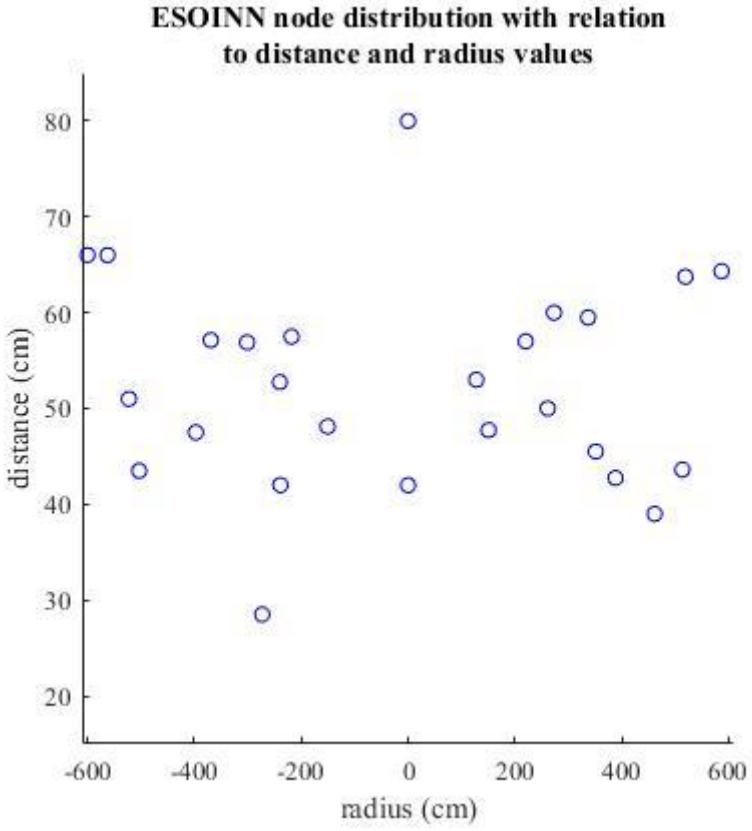


Fig. 4.13 - ESOINN node distribution with relation to distance and radius values of nodes

As seen in Fig. 4.13, the ESOINN after the first training, had 26 nodes. After this, a path following exercise was simulated with 3 different paths that were shown in Fig. 4.9 that the robot had to follow, and the outputs were obtained by implementing the IDW-ESOINN algorithm. This led to the creation of various virtual nodes so that

motions that were not contained in the current nodes could be made possible. The new temporary node distribution of the existing and virtual nodes is shown in Fig. 4.14.

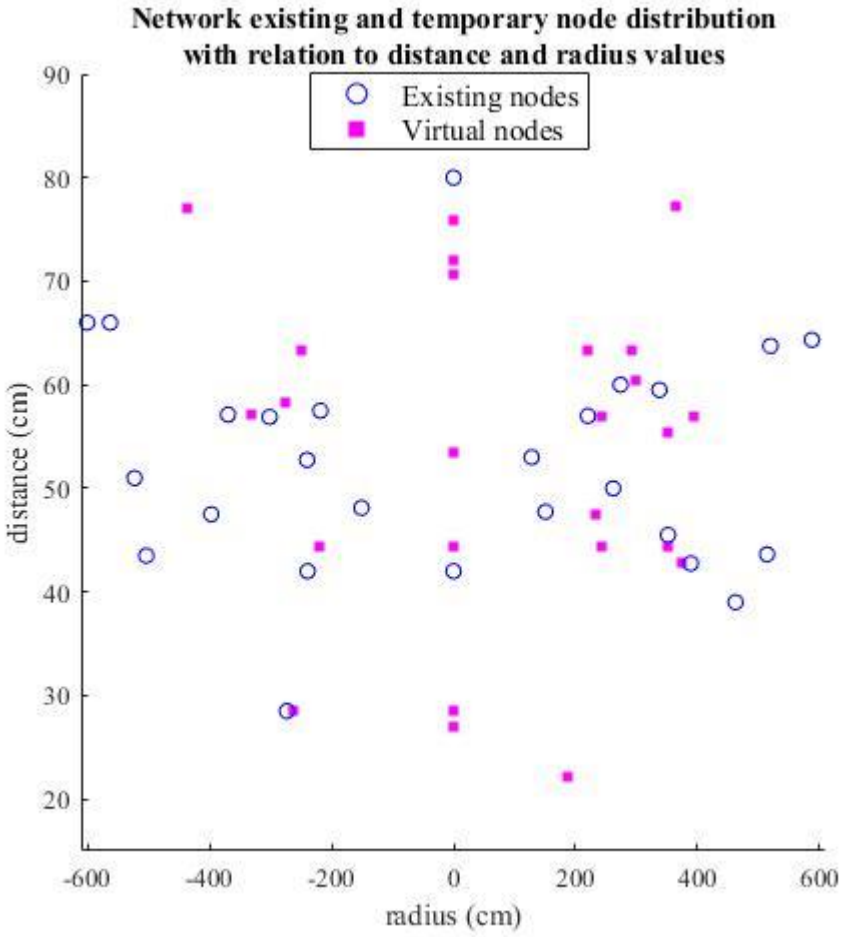


Fig. 4.14 - Temporary distribution of existing and virtual nodes after using the IDW-ESOINN for outputs

As seen in Fig. 4.14, by using the IDW-ESOINN algorithm, 25 additional virtual nodes (squares) were created to follow the simulated paths. This obviously shows that the resolution of the network directly increased after the application of the proposed method, as more motions were made possible.

(C) Distribution of Learnt Nodes after Incremental Learning

The next step taken was to investigate if these temporary virtual nodes led to a better distributed network after one step of incremental training. The initially trained network was run on 3 different paths as mentioned in the previous section, and outputs were obtained by using the ESOINN method and the IDW-ESOINN method. The two sets of outputs were then added separately to the initial network to get 2 completely new networks; one that was incremented using only the outputs from the ESOINN method, and the other that was trained using the virtual nodes created from the IDW-ESOINN method. Fig. 4.15 shows the node distribution for each of these networks before and after the incremental training step.

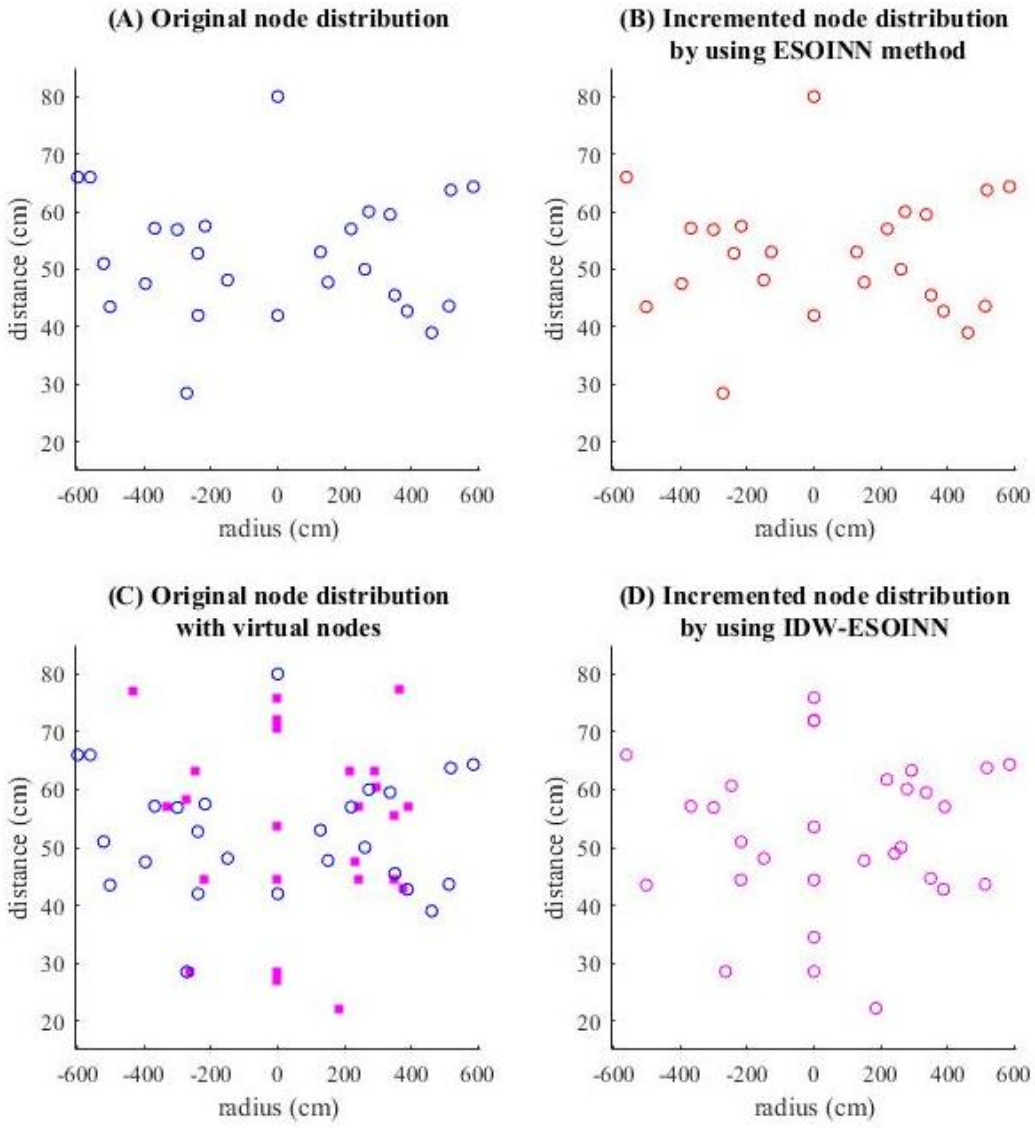


Fig. 4.15 - Network node distribution before and after incremental training

The node distribution as shown in Fig. 4.15(A), which was the original node distribution, changed to that of Fig. 4.15(B) when incremented by using outputs from only the ESOINN method. The number of nodes had only increased by 1 and the distribution looked almost the same, which shows that the effect of the incremental training was not very significant except for re-enforcing the existing nodes. The node distribution as shown in Fig. 4.15(C), which was the original node distribution along with the virtual nodes produced by the IDW-ESOINN method, changed to that of Fig. 4.15(D) when incremented. When considering Fig. 4.15(D), the number of nodes had increased from 26 to 33 as compared with the original network without virtual nodes in Fig. 4.15(A), and the distribution clearly had a different shape. If comparing Fig. 4.15(C) with Fig. 4.15(D), the number of nodes had fallen quite a bit, from 51 to 33, but this could be explained as during the learning process virtual nodes that occur only once would be removed as noise, or those that were quite close to already existing nodes would have been absorbed and the existing nodes would shift in their position.

To better compare the 2 new networks, they have been displayed together so that the spread can be seen more clearly in Fig. 4.16.

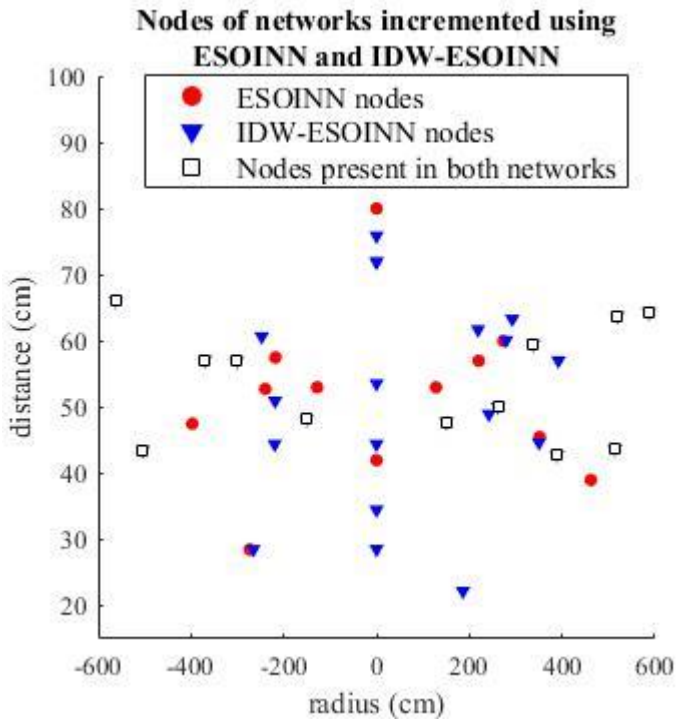


Fig. 4.16 - Distribution of nodes of network incremented using ESOINN and the IDW-ESOINN method

In Fig. 4.16, the red dots indicate nodes present in the network incremented using ESOINN, blue triangles indicate nodes in the IDW-ESOINN incremented network and the black squares indicate nodes that are present in both networks. It can be clearly seen that the blue triangles are more spread out in the given region, as compared to the red dots. This shows that by using the IDW-ESOINN method, the number of nodes not only increased, but also spread to areas that were not previously occupied, nor would be occupied if the nearest neighbouring node method was used to extract outputs.

4.3.3 Performance on Path Following

Once the incremented networks were obtained, the accuracy of the networks in following a simulated path had to be investigated, to check on the effects of the incremental training on the accuracy of the path following, which was crucial for this project. Initially simulations were conducted to compare the IDW-ESOINN method with the original ESOINN. Next, the proposed method was compared with the Kalman filtering method [96] on a number of simulations to prove its functionality in comparison to an already established method in trajectory control. Finally, a real time test was conducted to compare the performance of ESOINN and IDW-ESOINN on actual application on the robot.

(A) Performance Comparison on Simulated Path Following

The ESOINN and IDW-ESOINN used in this section were the networks created in the previous section (4.3.2) that was right after incrementation by using values from the 3 different path after initial training. The simulation was conducted by extracting the distance and radius values of each segment of a given path and feeding them into the networks. The outputs were obtained by finding the nearest neighbouring node to the distance and radius of the path segment. Interpolation was not done at this point as for this section the impact of the node distribution after incremental training was to be assessed and to observe if the network trained using the IDW-ESOINN method functioned better as compared to the original. Fig. 4.17 shows the original path to follow and the paths that were output as a result of the networks trying to follow the path.

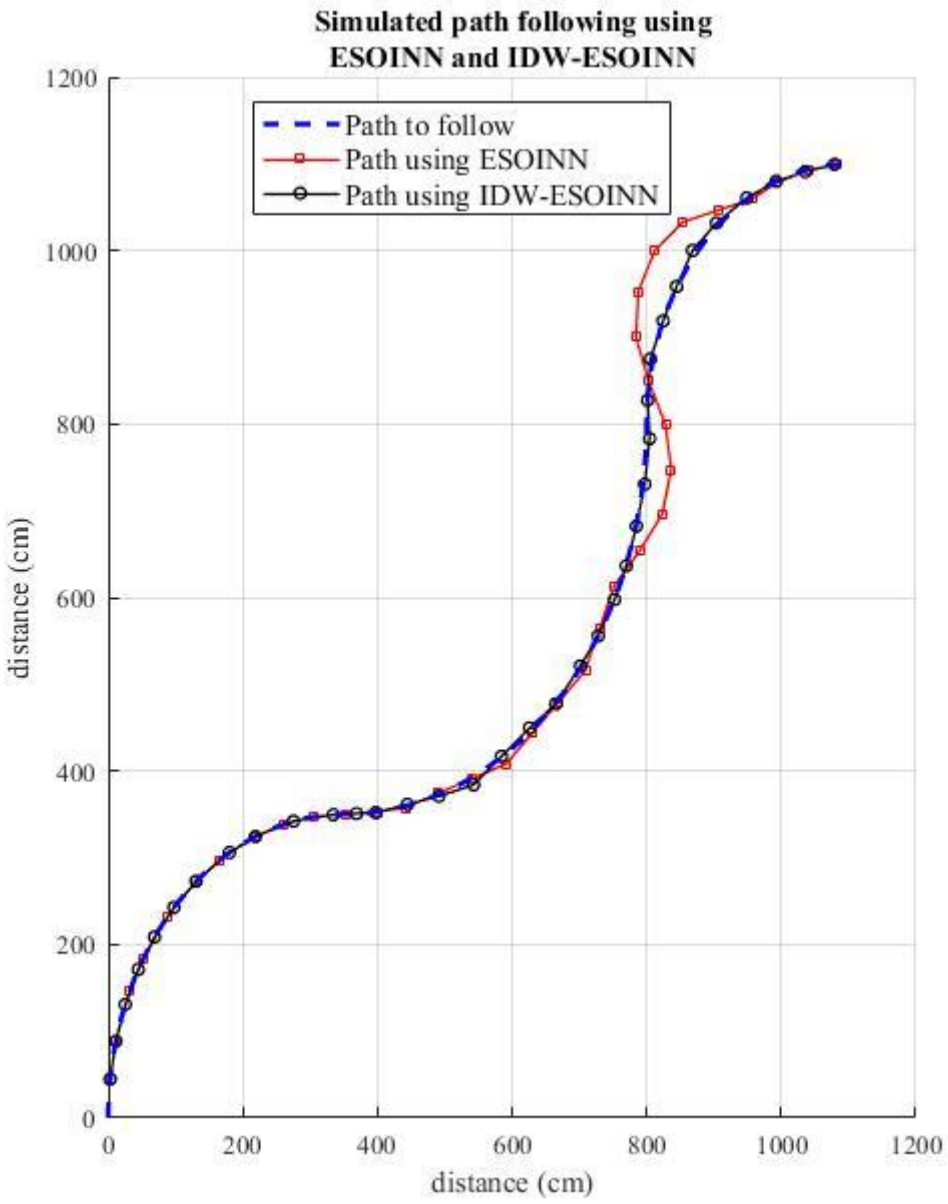


Fig. 4.17 - Simulated path following by the ESOINN and the IDW-ESOINN method

As seen in Fig. 4.17, the path travelled using the IDW-ESOINN method (black line with circles) was extremely similar to the original path to follow (blue dashed line), while the path travelled by using the ESOINN (red line with squares) had quite a few deviations along the path. The major deviations were caused by the fact that the ESOINN did not have any nodes that could perfectly match the motion needed, thus forcing the path to change to one that it could follow. This issue was not faced with the IDW-ESOINN method, as the node distribution enabled the path to have nodes that matched the needed movements more closely. The error between the 2 paths are listed in Table 4.11.

Table 4.11 - RMSE of simulated path travelled by using ESOINN and IDW-ESOINN

	ESOINN	**IDW-ESOINN**	**Error reduction (%)**
RMSE (x-axis) (cm)	18.124	9.737	46.3
RMSE (y-axis) (cm)	12.369	8.781	29.0

The values clearly show that the IDW-ESOINN outperformed the ESOINN method by around 9cm on the x-axis which was a reduction of 46.3% and about 4cm on the y-axis which was a reduction of 29.0%. Although both methods did have errors in following the path, the significantly reduced error by using the proposed method, validated its use.

To further evaluate the use of the IDW-ESOINN method, a comparison was made between the simulated path following done by the proposed method as well as a system that used a Kalman filter [96] to predict the trajectory of the robot. The Kalman Filter was trained by using the physically measured values from the data collection stage and predicted values of distance and radius obtained via encoder measurements to obtain a Kalman Gain of 0.012. The Kalman Gain was used to convert the distance and radius values from the path segments along with the table of measured values to predict the distance and radius the robot would follow. A total of 5 paths (including 3 of the previous paths) were created for testing as shown in Fig. 4.18 and were simulated to test the methods. As mentioned earlier, all the paths were created to ensure that the various motions of the robot can be tested to ensure it would perform accurately. Based on the literature, the number of paths vary depending on the case, ranging from 2-5 paths, as long as they sufficiently tested the functionality of the general motions to be performed by the robot [73], [78], [124]. The resulting RMSE from the outputs are shown in Table 4.12.

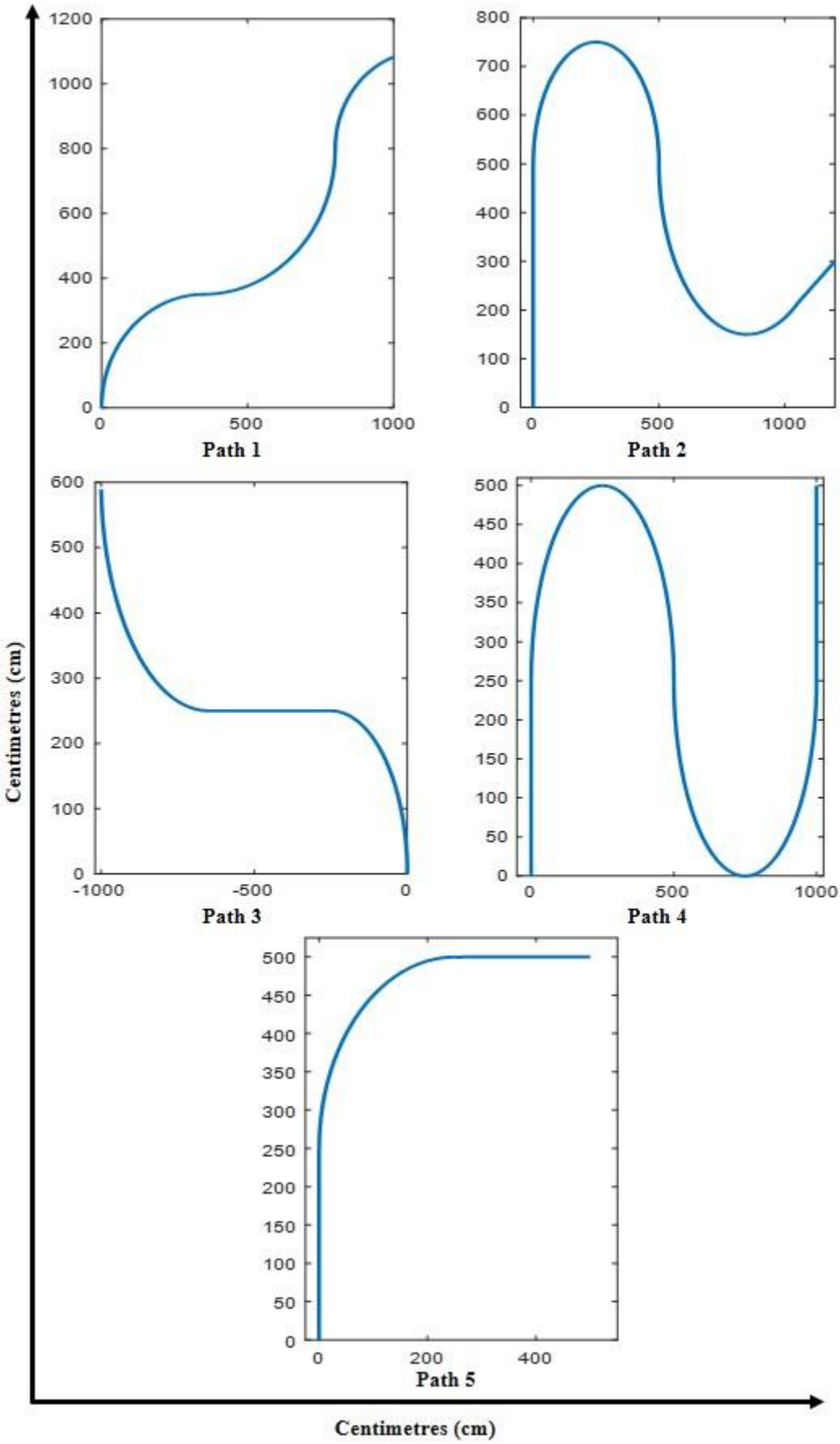


Fig. 4.18 - The 5 created paths for simulated testing

Table 4.12 - RMSE of simulated path travelled of 5 different paths

	RMSE value (x-axis : y-axis) (cm)		
Path	**ESOINN**	**IDW-ESOINN**	**Kalman filter**
1	18.124 : 12.369	9.737 : 8.781	7.242 : 8.916
2	7.617 : 7.601	6.953 : 7.338	9.726 : 8.131
3	7.744 : 7.937	8.165 : 7.864	17.003 : 6.350
4	6.562 : 7.536	2.950 : 4.773	6.349 : 8.181
5	9.249 : 3.331	6.543 : 3.011	15.238 : 5.066

All the simulations were conducted with the same network and Kalman Gain without any incrementation or update happening in between so as to have a better comparison between the methods. The method which had the lowest average RMSE values for each simulated path was highlighted in the table. The IDW-ESOINN method managed to obtain the lowest RMSE value for 3 of the 5 paths, and though it was higher for the other 2 paths, the RMSE value was extremely close to the lowest RMSE for that path. It also outperformed both the Kalman filtering method and the original ESOINN for 4 out of the 5 paths when compared individually. This shows that the IDW-ESOINN method had exceptional performance accuracy with a high degree of robustness in comparison to the other tested methods. Also, as compared to the Kalman filtering method, the proposed method did not require a full system model as long as prior data for training is obtained, which made the application of the method much simpler.

(B) Performance Comparison on Real-Time Path Following

Once simulations were completed, real-time testing was done on various paths to ensure the performance of the IDW-ESOINN algorithm. The ESOINN and IDW-ESOINN networks used in this section were the same ones used for the simulated tests without any changes. A path was generated and was fed into the trajectory controller. The

controller then followed a step-by-step approach to successfully navigate the given path. From the robot's starting point, a path segment in the range of the maximum possible distance that could be covered by the robot in one movement was segmented out. The distance and radius for that path segment was then extracted and fed into the network. The corresponding PWM signals to follow the given path segment was extracted from the network and the robot was moved. Once the robot came to a stop, the current position of the robot was updated by using the feedback information from the encoders and IMU. The next path segment that the robot could travel to was extracted and the steps above were followed continuously until the end of the path was reached. An example of the real time testing done is shown in Fig. 4.19.

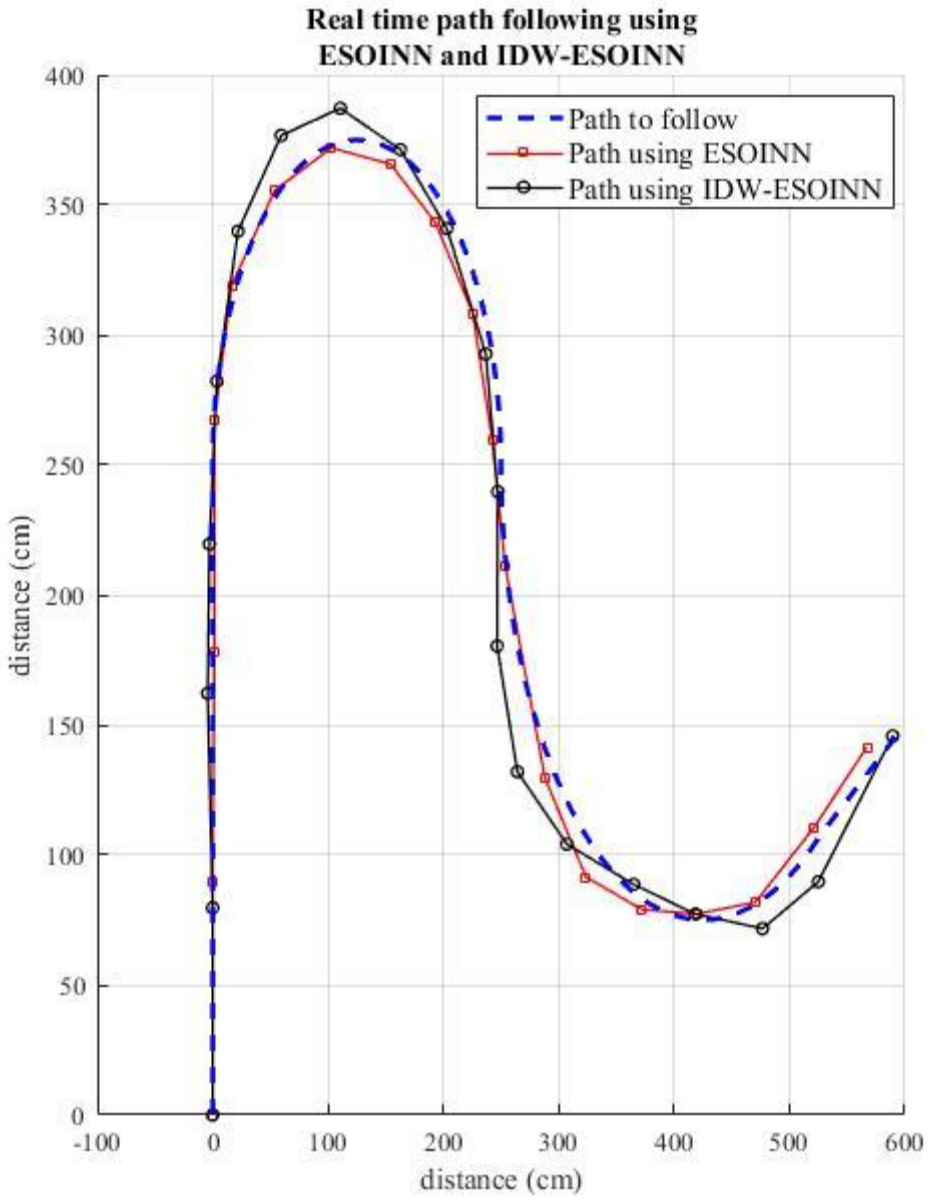


Fig. 4.19 - Real-time path following by using ESOINN and IDW-ESOINN

As seen in the figure, both paths travelled by using the ESOINN (red with squares) and the IDW-ESOINN method (black with circles) had deviations as compared to the original path (blue dashed line). The deviations were expected as unlike in a simulated environment, external factors such as terrain roughness, friction, wheel slip, momentum, etc. affected the motion of the robot, and so the outputs were not exactly as learnt by the networks. Though over time, these errors would reduce as the networks undergo more stages of incremental learning. The RMSE was calculated for both paths in terms of the error between the point the robot was supposed to travel to and the actual point it travelled to. The errors are shown in Table 4.13

Table 4.13 - RMSE of real time path travelled by ESOINN and IDW-ESOINN

	ESOINN	**IDW-ESOINN**	**Error reduction (%)**
RMSE (x-axis) (cm)	14.607	13.692	6.26
RMSE (y-axis) (cm)	10.073	9.551	5.19

The results in Table 4.13 still follow the same convention as that of Table 4.11 for simulated path following as the proposed method outputs were more accurate than the outputs from the ESOINN. The ESOINN appeared to perform better by just a visual estimation in Fig. 4.19, but it had higher inaccuracies as compared to the IDW-ESOINN when it came to following the motion it was asked to perform. The error reduction by using the IDW-ESOINN method was considerably lesser than that obtained in the simulated results, but this was expected as real time results usually differ from simulated ones. The results still do indicate that the proposed method outperformed the original ESOINN in this application in trajectory control.

CHAPTER 5: CONCLUSION

This project was designed to provide a solution to the decreasing amounts of labour in oil-palm plantations by creating a robot navigation system that was able to autonomously navigate through an oil-palm plantation. A robot was designed and constructed so that the created navigation system could be tested in a plantation environment. The navigation system was split into 3 major subsystems which were integrated together to form the whole navigation system.

The tree trunk detection system was constructed by making use of the Viola and Jones cascade detector with the addition of a novel pre-processing system to create a tree trunk detection method dubbed as the Oil Palm Tree Detector (OPTD). In detecting oil palm trees, the OPTD was second only to the Viola and Jones detector in detection rates for static images, while it had the lowest false positive rate even when compared with FNN, SVM and CNN based detectors. The computation time per image was reduced by the addition of the pre-processing method of the OPTD, as compared to using only the Viola and Jones detector. The OPTD method also increased the Viola and Jones detector accuracy in real time detection. With the inclusion of depth data by the use of the KINECT sensor, detection accuracy of the OPTD was increased by 9% in times / areas of low light and had a 100% accuracy at night with only depth data being used. It was also proven that the OPTD had a minimum detection rate of 91% while the robot was moving during the day when only colour images were present.

The proposed path planning system worked reasonably well in both simulated and real world testing. The path planning algorithm was able to successfully plan a new path whenever a new obstacle was encountered or when the robot went off track of the planned path. Though there was no novelty introduced in this area of the navigation system, the overall performance of the system proved that it was able to function successfully in a plantation environment.

For the trajectory control of the rover, an alternative control method was sought that was able to adapt to varied motions that had to be performed. The ESOINN was chosen as the controller, due to it being an unsupervised learning algorithm and its capability

of performing incremental learning while eliminating noise. The ESOINN proved to be a better learner than SOM, K-means and an adaptive NN, showing higher output accuracies during path following than the other 3 methods after each successive iteration of incremental learning, while also having an extremely short processing time for learning. Due to the ESOINN being unable to produce continuous outputs, a novel method (IDW-ESOINN) was proposed by using IDW interpolation on the nodes to resolve the issue. This resulted in obtaining continuous outputs via interpolation, leading to the creation of temporary virtual nodes, which were then used to create permanent nodes at newer areas in the next incremental learning phase. When compared with the ESOINN, the IDW-ESOINN method gave a higher number of nodes after incremental learning with a greater spread, thereby directly increasing the possible motions of the robot. In simulated testing of path following, an error reduction of 46.3% in the x-axis and 29% in the y-axis was obtained by using the proposed method as compared to the original ESOINN. In addition to this, a total of 5 paths were simulated, and the IDW-ESOINN method outperformed both the original ESOINN and the Kalman filter for 4 of those paths when individually compared. The high performance of the IDW-ESOINN was consistent throughout, with the RMSE always being either the lowest, or extremely close to the lowest result produced by the other methods. In real time results, the error difference between the original ESOINN and the proposed method was lowered to 6.26% and 5.19% respectively, proving that the IDW-ESOINN method performed with a higher accuracy than the original. By observing all the results mentioned, it can be said that the introduction of the IDW method to extract inter-node outputs, helped not only in the immediate extraction of outputs, but also in the next learning phase to increase the accuracy of the network.

All the results produced by simulations as well as the real time results mentioned in this thesis, proved that the entire navigation system performed extremely well in handling all the essential parts that it was created for. Within a range of 2-6m of a tree, the OPTD method would be able to detect and narrow down on the tree with a high accuracy, leading to a path being planned and the IDW-ESOINN moving the robot accurately to the tree itself. All subsystems performed as per their requirements, showing robustness in the output data, which ensures the system’s performance in the varied terrain of an oil palm plantation.

REFERENCES

[1] A. Raghu, "Labour crunch hurts Malaysian palm oil growers as Indonesians stay home," *Reuters*, 2014. [Online]. Available: https://www.reuters.com/article/us-palmoil-labour/labour-crunch-hurts-malaysian-palm-oil-growers-as-indonesians-stay-home-idUSBREA3Q0P320140427.

[2] Bernama, "Plantation industry hopes new minister will address critical issues," *astro AWANI*, 2016. [Online]. Available: http://english.astroawani.com/malaysia-news/plantation-industry-hopes-new-minister-will-address-critical-issues-109382.

[3] E. Chow, "Malaysia palm planters face labor shortage as Indonesia workers stay away," *Reuters*, 2017. [Online]. Available: https://www.reuters.com/article/us-malaysia-palmoil-labour-idUSKBN1790VO.

[4] L. Shao, Y. Mu, J. Liu, G. Dong, H. Liu, and P. Guo, "The trunk of the image recognition based on BP neural network," in *2014 IEEE International Conference on Mechatronics and Automation*, 2014, vol. 53, no. 9, pp. 1800–1805.

[5] A. Jaakkola, J. Hyyppä, A. Kukko, X. Yu, H. Kaartinen, M. Lehtomäki, and Y. Lin, "A low-cost multi-sensoral mobile mapping system and its feasibility for tree measurements," *ISPRS J. Photogramm. Remote Sens.*, vol. 65, no. 6, pp. 514–522, Nov. 2010.

[6] A. Bechar and C. Vigneault, "Agricultural robots for field operations. Part 2: Operations and systems," *Biosyst. Eng.*, vol. 153, pp. 110–128, Jan. 2017.

[7] Y. Lu and C. Rasmussen, "Tree trunk detection using contrast templates," in *2011 18th IEEE International Conference on Image Processing*, 2011, pp. 1253–1256.

[8] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1, pp. 511–518, 2001.

[9] X. Sun, W. Yeoh, and S. Koenig, "Moving Target D* Lite," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, 2010, pp. 67–74.

[10] S. Furao, T. Ogura, and O. Hasegawa, "An enhanced self-organizing incremental neural network for online unsupervised learning," *Neural Networks*, vol. 20, no.

8, pp. 893–903, Oct. 2007.

[11] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data," in *Proceedings of the 1968 23rd ACM national conference on* -, 1968, pp. 517–524.

[12] M. Z. Abdullah, L. C. Guan, K. C. Lim, and A. A. Karim, "The applications of computer vision system and tomographic radar imaging for assessing physical properties of food," *J. Food Eng.*, vol. 61, no. 1 SPEC., pp. 125–135, 2004.

[13] M. Almonacid, R. J. Saltarén, R. Aracil, and O. Reinoso, "Motion planning of a climbing parallel robot," *IEEE Trans. Robot. Autom.*, vol. 19, no. 3, pp. 485–489, 2003.

[14] W. I. W. Ismail, "Research and Development of Oil Palm Harvester Robot at Universiti Putra Malaysia," *Int. J. Eng. Technol.*, vol. 7, no. 2, pp. 87–94, 2010.

[15] L. Shao, X. Chen, B. Milne, and P. Guo, "A novel tree trunk recognition approach for forestry harvesting robot," in *2014 9th IEEE Conference on Industrial Electronics and Applications*, 2014, pp. 862–866.

[16] A. Benmakhlouf and A. Louchene, "Simple Fuzzy Logic Based Path Tracking Controller for a Mobile Robot," *J. Electr. Eng.*, pp. 1–8.

[17] N. B. Hoang and H. J. Kang, "Neural network-based adaptive tracking control of mobile robots in the presence of wheel slip and external disturbance force," *Neurocomputing*, vol. 188, pp. 12–22, 2016.

[18] R. Cui, C. Yang, Y. Li, and S. Sharma, "Adaptive Neural Network Control of AUVs With Control Input Nonlinearities Using Reinforcement Learning," *IEEE Trans. Syst. Man, Cybern. Syst.*, vol. 47, no. 6, pp. 1019–1029, Jun. 2017.

[19] C. Sun, W. He, W. Ge, and C. Chang, "Adaptive Neural Network Control of Biped Robots," *IEEE Trans. Syst. Man, Cybern. Syst.*, vol. 47, no. 2, pp. 1–12, 2016.

[20] K. Tanigaki, T. Fujiura, A. Akase, and J. Imagawa, "Cherry-harvesting robot," *Comput. Electron. Agric.*, vol. 63, no. 1, pp. 65–72, 2008.

[21] Z. De-An, L. Jidong, J. Wei, Z. Ying, and C. Yu, "Design and control of an apple harvesting robot," *Biosyst. Eng.*, vol. 110, no. 2, pp. 112–122, 2011.

[22] A. Silwal, J. R. Davidson, M. Karkee, C. Mo, Q. Zhang, and K. Lewis, "Design, integration, and field evaluation of a robotic apple harvester," *J. F. Robot.*, vol. 34, no. 6, pp. 1140–1159, Sep. 2017.

[23] J. T. Colmenero-Martinez, G. L. Blanco-Roldán, S. Bayano-Tejero, F. J.

Castillo-Ruiz, R. R. Sola-Guirado, and J. A. Gil-Ribes, “An automatic trunk-detection system for intensive olive harvesting with trunk shaker,” *Biosyst. Eng.*, vol. 172, pp. 92–101, Aug. 2018.

[24] T. Bakker, K. van Asselt, J. Bontsema, J. Müller, and G. van Straten, “Autonomous navigation using a robot platform in a sugar beet field,” *Biosyst. Eng.*, vol. 109, no. 4, pp. 357–368, 2011.

[25] J. Xue, L. Zhang, and T. E. Grift, “Variable field-of-view machine vision based row guidance of an agricultural robot,” *Comput. Electron. Agric.*, vol. 84, pp. 85–91, 2012.

[26] S. a. Hiremath, G. W. a M. van der Heijden, F. K. van Evert, A. Stein, and C. J. F. Ter Braak, “Laser range finder model for autonomous navigation of a robot in a maize field using a particle filter,” *Comput. Electron. Agric.*, vol. 100, pp. 41–50, 2014.

[27] E. Kayacan, H. Ramon, O. Kaynak, and W. Saeys, “Towards Agrobots: Trajectory Control of an Autonomous Tractor Using Type-2 Fuzzy Logic Controllers,” *IEEE Trans. Mechatronics*, vol. 20, no. 1, pp. 1–12, 2013.

[28] X. Yin and N. Noguchi, “Development of a target following system for a field robot,” *Comput. Electron. Agric.*, vol. 4, no. PART 1, pp. 17–24, 2013.

[29] R. Vidoni, M. Bietresato, A. Gasparetto, and F. Mazzetto, “Evaluation and stability comparison of different vehicle configurations for robotic agricultural operations on side-slopes,” *Biosyst. Eng.*, vol. 129, pp. 197–211, 2015.

[30] J. M. Ortiz and M. Olivares, “A Vision Based Navigation System for an Agricultural Field Robot,” in *2006 IEEE 3rd Latin American Robotics Symposium*, 2006, pp. 106–114.

[31] F. A. Auat Cheein and R. Carelli, “Agricultural robotics: Unmanned robotic service units in agricultural tasks,” *IEEE Ind. Electron. Mag.*, vol. 7, no. 3, pp. 48–58, 2013.

[32] S. S. H. Hajjaj and K. S. M. Sahari, “Review of agriculture robotics: Practicality and feasibility,” in *2016 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*, 2016, pp. 194–198.

[33] M. Makky and P. Soni, “Development of an automatic grading machine for oil palm fresh fruits bunches (FFBs) based on machine vision,” *Comput. Electron. Agric.*, vol. 93, pp. 129–139, Apr. 2013.

[34] M. S. M. Kassim, W. I. W. Ismail, A. R. Ramli, and S. K. Bejo, “Image

Clustering Technique in Oil Palm Fresh Fruit Bunch (FFB) Growth Modeling," *Agric. Agric. Sci. Procedia*, vol. 2, pp. 337–344, Jan. 2014.

[35] M. Makky, "A Portable Low-cost Non-destructive Ripeness Inspection for Oil Palm FFB," *Agric. Agric. Sci. Procedia*, vol. 9, pp. 230–240, Jan. 2016.

[36] B. Pamornnak, S. Limsiroratana, T. Khaorapapong, M. Chongcheawchamnan, and A. Ruckelshausen, "An automatic and rapid system for grading palm bunch using a Kinect camera," *Comput. Electron. Agric.*, vol. 143, pp. 227–237, Dec. 2017.

[37] M. Vidal and J. M. Amigo, "Pre-processing of hyperspectral images. Essential steps before image analysis," *Chemom. Intell. Lab. Syst.*, vol. 117, pp. 138–148, 2012.

[38] Y. Hongbo and H. Xia, "Histogram modification using grey-level co-occurrence matrix for image contrast enhancement," *IET Image Process.*, vol. 8, no. 12, pp. 782–793, 2014.

[39] H. Han, S. Shan, X. Chen, and W. Gao, "A comparative study on illumination preprocessing in face recognition," *Pattern Recognit.*, vol. 46, no. 6, pp. 1691–1699, 2013.

[40] a. K. Bhandari, V. Soni, a. Kumar, and G. K. Singh, "Cuckoo search algorithm based satellite image contrast and brightness enhancement using DWT-SVD," *ISA Trans.*, vol. 53, no. 4, pp. 1286–1296, 2014.

[41] Z.-H. Huang, W.-J. Li, J. Shang, J. Wang, and T. Zhang, "Non-uniform patch based face recognition via 2D-DWT," *Image Vis. Comput.*, vol. 37, pp. 12–19, 2015.

[42] Y. Gurwicz, R. Yehezkel, and B. Lachover, "Multiclass object classification for real-time video surveillance systems," *Pattern Recognit. Lett.*, vol. 32, no. 6, pp. 805–815, 2011.

[43] Y. Huang, W. Lu, W. Sun, and D. Long, "Improved DCT-based detection of copy-move forgery in images," *Forensic Sci. Int.*, vol. 206, no. 1–3, pp. 178–184, 2011.

[44] R. Atta and M. Ghanbari, "Low-contrast satellite images enhancement using discrete cosine transform pyramid and singular value decomposition," *IET Image Process.*, vol. 7, no. April, pp. 472–483, 2013.

[45] L. I. Kuncheva and W. J. Faithfull, "PCA feature extraction for change detection in multidimensional unlabeled data," *IEEE Trans. Neural Networks Learn. Syst.*,

vol. 25, no. 1, pp. 69–80, 2014.

[46] S.-K. Oh, S.-H. Yoo, and W. Pedrycz, "Design of face recognition algorithm using PCA -LDA combined for hybrid data pre-processing and polynomial-based RBF neural networks : Design and its application," *Expert Syst. Appl.*, vol. 40, no. 5, pp. 1451–1466, 2012.

[47] Y. Gao, C. Spiteri, M. T. Pham, and S. Al-Milli, "A survey on recent object detection techniques useful for monocular vision-based planetary terrain classification," *Rob. Auton. Syst.*, vol. 62, no. 2, pp. 151–167, 2014.

[48] B. Widrob, B. Widrob, M. A. Lehr, M. A. Lehr, B. Widrow, and M. A. Lehr, "30 years of adaptive neural networks: perceptron, Madaline, andbackpropagation," *Proc. IEEE*, vol. 78, no. 9, pp. 1415–1442, 1990.

[49] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1. pp. 318–362, 1986.

[50] R. Kouskouridas, A. Gasteratos, and C. Emmanouilidis, "Efficient representation and feature extraction for neural network-based 3D object pose estimation," *Neurocomputing*, vol. 120, pp. 90–100, 2013.

[51] J. Moody and C. J. Darken, "Fast Learning in Networks of Locally-Tuned Processing Units," *Neural Computation*, vol. 1, no. 2. pp. 281–294, 1989.

[52] S.-C. Huang and B.-H. Do, "Radial Basis Function Based Neural Network for Motion Detection in Dynamic Scenes.," *IEEE Trans. Cybern.*, vol. 44, no. 1, pp. 114–125, 2013.

[53] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, 2012, pp. 1097–1105.

[54] H. Zhou, C. Yan, and H. Huang, "Tree Species Identification Based on Convolutional Neural Networks," in *2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, 2016, pp. 103–106.

[55] A. P, H. MP, H. Sounder, N. K, V. P V, and R. Hebbar, "CNN Based Technique for Automatic Tree Counting Using Very High Resolution Data," in *2018 International Conference on Design Innovations for 3Cs Compute Communicate Control (ICDI3C)*, 2018, pp. 127–129.

[56] L. Fu, Y. Feng, Y. Majeed, X. Zhang, J. Zhang, M. Karkee, and Q. Zhang, "Kiwifruit detection in field images using Faster R-CNN with ZFNet," *IFAC-PapersOnLine*, vol. 51, no. 17, pp. 45–50, Jan. 2018.

[57] S. Branson, J. D. Wegner, D. Hall, N. Lang, K. Schindler, and P. Perona, "From Google Maps to a fine-grained catalog of street trees," *ISPRS J. Photogramm. Remote Sens.*, vol. 135, pp. 13–30, Jan. 2018.

[58] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[59] Y. Pang, K. Zhang, Y. Yuan, and K. Wang, "Distributed Object Detection With Linear SVMs," *IEEE Trans. Cybern.*, vol. 44, no. 11, pp. 1–12, Nov. 2014.

[60] F. Kurtulmuş and I. Kavdir, "Detecting corn tassels using computer vision and support vector machines," *Expert Syst. Appl.*, vol. 41, no. 16, pp. 7390–7397, 2014.

[61] A. Borji, "What is a Salient Object? A Dataset and a Baseline Model for Salient Object Detection," *IEEE Trans. Image Process.*, vol. 24, no. 2, pp. 742–756, 2015.

[62] R. Shi, Z. Liu, H. Du, X. Zhang, and L. Shen, "Region diversity maximization for salient object detection," *IEEE Signal Process. Lett.*, vol. 19, no. 4, pp. 215–218, 2012.

[63] Y. Freund and R. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," *Comput. Learn. theory*, vol. 55, no. 1, pp. 119–139, 1995.

[64] R. Lienhart and J. Maydt, "An extended set of Haar-like features for rapid object detection," *Proceedings. Int. Conf. Image Process.*, vol. 1, pp. 900–903, 2002.

[65] S. Vural, Y. Mae, H. Uvet, and T. Arai, "Multi-view fast object detection by using extended haar filters in uncontrolled environments," *Pattern Recognit. Lett.*, vol. 33, no. 2, pp. 126–133, 2012.

[66] K. Y. Park and S. Y. Hwang, "An improved Haar-like feature for efficient object detection," *Pattern Recognit. Lett.*, vol. 42, no. 1, pp. 148–153, 2014.

[67] D. H. Kim, S. U. Jung, and M. J. Chung, "Extension of cascaded simple feature based face detection to facial expression recognition," *Pattern Recognit. Lett.*, vol. 29, no. 11, pp. 1621–1631, 2008.

[68] V. D. Hoang, M. H. Le, and K. H. Jo, "Hybrid cascade boosting machine using variant scale blocks based HOG features for pedestrian detection,"

Neurocomputing, vol. 135, pp. 357–366, 2014.

[69] G.-C. C. Chen and C.-F. F. Juang, "Object detection using color entropies and a fuzzy classifier," *IEEE Comput. Intell. Mag.*, vol. 8, no. 1, pp. 33–45, 2013.

[70] P. M. Baggenstoss, S. Madhogaria, M. Schikora, W. Koch, and D. Cremers, "Car detection in aerial images by fusion of HOG and a causal MRF," in *IEEE Transactions on Aerospace and Electronic Systems*, 2015, vol. 51, no. 1, pp. 575–590.

[71] P. Shivakumara, R. P. Sreedhar, T. Q. Phan, S. Lu, and C. L. Tan, "Multioriented video scene text detection through bayesian classification and boundary growing," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 8, pp. 1227–1235, 2012.

[72] F. Zaklouta and B. Stanciulescu, "Real-time traffic-sign recognition using tree classifiers," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 4, pp. 1507–1514, 2012.

[73] G. Bayar, M. Bergerman, A. B. Koku, and E. I. Konukseven, "Localization and control of an autonomous orchard vehicle," *Comput. Electron. Agric.*, vol. 115, pp. 118–128, Jul. 2015.

[74] J. Gimenez, D. Herrera, S. Tosetti, and R. Carelli, "Optimization methodology to fruit grove mapping in precision agriculture," *Comput. Electron. Agric.*, vol. 116, pp. 88–100, 2015.

[75] R. Sanz, J. R. Rosell, J. Llorens, E. Gil, and S. Planas, "Relationship between tree row LIDAR-volume and leaf area density for fruit orchards and vineyards obtained with a LIDAR 3D Dynamic Measurement System," *Agric. For. Meteorol.*, vol. 171–172, pp. 153–162, 2013.

[76] J. R. Rosell, J. Llorens, R. Sanz, J. Arnó, M. Ribes-Dasi, J. Masip, A. Escolà, F. Camp, F. Solanelles, F. Gràcia, E. Gil, L. Val, S. Planas, and J. Palacín, "Obtaining the three-dimensional structure of tree orchards from remote 2D terrestrial LIDAR scanning," *Agric. For. Meteorol.*, vol. 149, no. 9, pp. 1505–1515, Sep. 2009.

[77] P. Forsman and A. Halme, "3-D mapping of natural environments with trees by means of mobile perception," *IEEE Trans. Robot.*, vol. 21, no. 3, pp. 482–490, 2005.

[78] J. Zhang, S. Maeta, M. Bergerman, and S. Singh, "Mapping Orchards for Autonomous Navigation," in *2014 ASABE Annual International Meeting*, 2014,

vol. 7004, pp. 1–9.

[79] F. Auat Cheein, G. Steiner, G. Perez Paina, and R. Carelli, "Optimized EIF-SLAM algorithm for precision agriculture mapping based on stems detection," *Comput. Electron. Agric.*, vol. 78, no. 2, pp. 195–207, 2011.

[80] J. Torres-Sospedra and P. Nebot, "Two-stage procedure based on smoothed ensembles of neural networks applied to weed detection in orange groves," *Biosyst. Eng.*, vol. 123, pp. 40–55, Jul. 2014.

[81] R. Zhong, J. Wei, W. Su, and Y. F. Chen, "A method for extracting trees from vehicle-borne laser scanning data," *Math. Comput. Model.*, vol. 58, no. 3–4, pp. 727–736, 2013.

[82] N. Shalal, T. Low, C. McCarthy, and N. Hancock, "Orchard mapping and mobile robot localisation using on-board camera and laser scanner data fusion - Part A: Tree detection," *Comput. Electron. Agric.*, vol. 119, pp. 254–266, 2015.

[83] N. Shalal, T. Low, C. McCarthy, and N. Hancock, "Orchard mapping and mobile robot localisation using on-board camera and laser scanner data fusion - Part B: Mapping and localisation," *Comput. Electron. Agric.*, vol. 119, pp. 267–278, 2015.

[84] X. Chen, S. Wang, B. Zhang, and L. Luo, "Multi-feature fusion tree trunk detection and orchard mobile robot localization using camera/ultrasonic sensors," *Comput. Electron. Agric.*, vol. 147, pp. 91–108, Apr. 2018.

[85] S. Nissimov, J. Goldberger, and V. Alchanatis, "Obstacle detection in a greenhouse environment using the Kinect sensor," *Comput. Electron. Agric.*, vol. 113, pp. 104–115, Apr. 2015.

[86] J. Hernandez-Aceituno, R. Arnay, J. Toledo, and L. Acosta, "Using Kinect on an Autonomous Vehicle for Outdoors Obstacle Detection," *IEEE Sens. J.*, vol. 16, no. 10, pp. 3603–3610, May 2016.

[87] Y. Majeed, J. Zhang, X. Zhang, L. Fu, M. Karkee, Q. Zhang, and M. D. Whiting, "Apple Tree Trunk and Branch Segmentation for Automatic Trellis Training Using Convolutional Neural Network Based Semantic Segmentation," *IFAC-PapersOnLine*, vol. 51, no. 17, pp. 75–80, Jan. 2018.

[88] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, 1959.

[89] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4,

no. 2, pp. 100–107, 1968.

[90] A. Stentz, "The Focussed D* Algorithm for Real-time Replanning," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, 1995, pp. 1652–1659.

[91] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong Planning A*," *Artif. Intell.*, vol. 155, no. 1–2, pp. 93–146, 2004.

[92] S. Koenig and M. Likhachev, "D* Lite," *Proc. Eighteenth Natl. Conf. Artif. Intell.*, pp. 476–483, 2002.

[93] J. Borenstein and Y. Koren, "Obstacle avoidance with ultrasonic sensors," *IEEE J. Robot. Autom.*, vol. 4, no. 2, pp. 213–218, Apr. 1988.

[94] M. Kang, K. Kim, D. Noh, J. Han, and S. Ko, "A robust obstacle detection method for robotic vacuum cleaners," *IEEE Trans. Consum. Electron.*, vol. 60, no. 4, pp. 587–595, Nov. 2014.

[95] F. N. Martins, W. C. Celeste, R. Carelli, M. Sarcinelli-Filho, and T. F. Bastos-Filho, "An adaptive dynamic controller for autonomous mobile robot trajectory tracking," *Control Eng. Pract.*, vol. 16, no. 11, pp. 1354–1363, Nov. 2008.

[96] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Trans. ASME--Journal Basic Eng.*, 1960.

[97] H. Chen, H. Lim, S. Hwang, and J. Lee, "Efficient driving of tracking robot using kalman filters," in *2013 13th International Conference on Control, Automation and Systems (ICCAS 2013)*, 2013, pp. 119–123.

[98] O. Koç, G. Maeda, and J. Peters, "Online optimal trajectory generation for robot table tennis," *Rob. Auton. Syst.*, vol. 105, pp. 121–137, Jul. 2018.

[99] Ngatini, E. Apriliani, and H. Nurhadi, "Ensemble and Fuzzy Kalman Filter for position estimation of an autonomous underwater vehicle based on dynamical system of AUV motion," *Expert Syst. Appl.*, vol. 68, pp. 29–35, Feb. 2017.

[100] Y. V Parkale, "Comparison of ANN Controller and PID Controller for Industrial Water Bath Temperature Control System using MATLAB Environment," *Int. J. Comput. Appl.*, vol. 53, no. 2, pp. 1–6, 2012.

[101] L. Caracciolo, A. de Luca, and S. Iannitti, "Trajectory tracking control of a four-wheel differentially driven mobile robot," *Proc. 1999 IEEE Int. Conf. Robot. Autom. (Cat. No.99CH36288C)*, vol. 4, no. May, pp. 2632–2638, 1999.

[102] K. Kozlowski and D. Pazderski, "Modeling and control of a 4-wheel skid-steering mobile robot," *Int. J. Appl. Math. Comput. Sci.*, vol. 14, no. 4, pp. 477–

496, 2004.

[103] P. R. Ouyang, V. Pano, and T. Dam, “PID contour tracking control in position domain,” in *IEEE International Symposium on Industrial Electronics*, 2012, pp. 1297–1302.

[104] G. M. Hoffmann, S. L. Waslander, and C. J. Tomlin, “Quadrotor Helicopter Trajectory Tracking Control,” *Electr. Eng.*, vol. 44, no. August, pp. 1–14, 2008.

[105] J. Yen, “Fuzzy logic - a modern perspective,” *IEEE Trans. Knowl. Data Eng.*, vol. 11, no. 1, pp. 153–165, 1999.

[106] H. Zhang, K. Yuan, and S. Mei, “Fuzzy Logic Cross-coupling Control of Wheeled Mobile Robots,” in *2006 International Conference on Mechatronics and Automation*, 2006, pp. 740–744.

[107] D. Chwa, “Fuzzy adaptive tracking control of wheeled mobile robots with state-dependent kinematic and dynamic disturbances,” *IEEE Trans. Fuzzy Syst.*, vol. 20, no. 3, pp. 587–593, Jun. 2012.

[108] M. Begnini, D. W. Bertol, and N. A. Martins, “A robust adaptive fuzzy variable structure tracking control for the wheeled mobile robot: Simulation and experimental results,” *Control Eng. Pract.*, vol. 64, no. January 2016, pp. 27–43, Jul. 2017.

[109] S. Ahmad and M. O. Tokhi, “Steering Motion Control Enhancement Scheme of Two Wheeled Wheelchair in Confined Spaces,” *Int. J. Autom. Control Eng.*, vol. 2, no. 4, pp. 179–189, 2013.

[110] P. Joshi and J. Valasek, “Direct comparison of neural network, fuzzy logic and model predictive variable structure vortex flow controllers,” in *Guidance, Navigation, and Control Conference and Exhibit*, 1999, no. c.

[111] K. Gurney, *An Introduction to Neural Networks*. Bristol, PA, USA: Taylor & Francis, Inc., 1997.

[112] S. Haykin, *Neural Networks and Learning Machines*, vol. 3. 2008.

[113] Z.-H. Jiang, “Trajectory Control of Robot Manipulators Using a Neural Network Controller,” in *Robot Manipulators Trends and Development*, InTech, 2010, pp. 361–377.

[114] Jizhong Xiao, Qing Song, and Danwei Wang, “A learning control scheme based on neural networks for repeatable robot trajectory tracking,” in *Proceedings of the 1999 IEEE International Symposium on Intelligent Control Intelligent Systems and Semiotics (Cat. No.99CH37014)*, 1999, pp. 102–107.

[115] E. D. Michie, D. J. Spiegelhalter, and C. C. Taylor, "Machine Learning , Neural and Statistical Classification," 1994.

[116] S. X. Yang and M. Meng, "An efficient neural network approach to dynamic robot motion planning.," *Neural Networks*, vol. 13, no. 2, pp. 143–148, 2000.

[117] M. Seyr, S. Jakubek, and G. Novak, *Neural Network Predictive Trajectory Tracking of an Autonomous Two-Wheeled Mobile Robot*, vol. 38, no. 1. IFAC, 2005.

[118] G. Yanfeng, Z. Hua, and Y. Yanhui, "Back-Stepping and Neural Network Control of a Mobile Robot for Curved Weld Seam Tracking," *Procedia Eng.*, vol. 15, pp. 38–44, 2011.

[119] R. Fierro and F. L. Lewis, "Control of a nonholonomic mobile robot using neural networks.," *IEEE Trans. Neural Netw.*, vol. 9, no. 4, pp. 589–600, Jul. 1998.

[120] M. K. Bugeja, S. G. Fabri, and L. Camilleri, "Dual Adaptive Dynamic Control of Mobile Robots Using Neural Networks," *IEEE Trans. Syst. Man, Cybern. Part B*, vol. 39, no. 1, pp. 129–141, Feb. 2009.

[121] D. Xu, D. Zhao, J. Yi, and X. Tan, "Trajectory tracking control of omnidirectional wheeled mobile manipulators: Robust neural network-based sliding mode approach," *IEEE Trans. Syst. Man, Cybern. Part B Cybern.*, vol. 39, no. 3, pp. 788–799, 2009.

[122] F. G. Rossomando, C. Soria, and R. Carelli, "Autonomous mobile robots navigation using RBF neural compensator," *Control Eng. Pract.*, vol. 19, no. 3, pp. 215–222, Mar. 2011.

[123] Z. Li, J. Deng, R. Lu, Y. Xu, J. Bai, and C.-Y. Su, "Trajectory-Tracking Control of Mobile Robot Systems Incorporating Neural-Dynamic Optimized Model Predictive Approach," *IEEE Trans. Syst. Man, Cybern. Syst.*, vol. 46, no. 6, pp. 740–749, Jun. 2016.

[124] H.-M. Yen, T.-H. S. Li, and Y.-C. Chang, "Design of a robust neural network-based tracking controller for a class of electrically driven nonholonomic mechanical systems," *Inf. Sci. (Ny).*, vol. 222, pp. 559–575, Feb. 2013.

[125] M. Li, Y. Li, S. S. Ge, and T. H. Lee, "Adaptive Control of Robotic Manipulators With Unified Motion Constraints," *IEEE Trans. Syst. Man, Cybern. Syst.*, pp. 1–11, 2016.

[126] T. Kohonen, "The self-organizing map," *Proc. IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[127] S. Furao and O. Hasegawa, "An incremental network for on-line unsupervised classification and topology learning," *Neural Networks*, vol. 19, no. 1, pp. 90–106, Jan. 2006.

[128] A. Kawewong, Y. Honda, M. Tsuboyama, and O. Hasegawa, "Common-patterns based mapping for robot navigation," in *2008 IEEE International Conference on Robotics and Biomimetics*, 2009, pp. 608–614.

[129] S. Tangruamsub, M. Tsuboyama, A. Kawewong, and O. Hasegawa, "Mobile robot vision-based navigation using self-organizing and incremental neural networks," in *2009 International Joint Conference on Neural Networks*, 2009, pp. 3094–3101.

[130] C. Yang, S. Kao, F. Lee, P. Hung, and K. E. Y. Words, "A Study of the 12 Different Interpolation Methods----A Case Study of."

[131] M. Aupetit, P. Couturier, and P. Massotte, "Function Approximation with Continuous Self-Organizing Maps using Neighboring Influence Interpolation," in *Proceedings of the International ICSC Symposium on Neural Computation*, 2000, no. May.

[132] A. M. Kalteh and R. Berndtsson, "Interpolating monthly precipitation by self-organizing map (SOM) and multilayer perceptron (MLP)," *Hydrol. Sci. J.*, vol. 52, no. 2, pp. 305–317, Apr. 2007.

[133] S. Conte and C. DeBoor, *Elementary Numerical Analysis*, Third Edit. McGraw Hills, 1980.

[134] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and other kernel based learning methods*, vol. 22, no. 2. 2000.

[135] M. T. Hagan and M. B. Menhaj, "Training Feedforward Networks with the Marquardt Algorithm," *IEEE Trans. Neural Networks*, vol. 5, no. 6, pp. 989–993, 1994.

[136] Nic Chin & Partners, "Daylight Harvesting In Malaysia." [Online]. Available: http://www.nicchinandpartners.com/green-engineering/daylight-harvesting/.

[137] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. 28, no. 2, pp. 129–137, Mar. 1982.