Autonomous System for Node and Edge Detection, Path Planning, and Efficient Colour-coded Box Handling in Complex Networks

Chetna Mundra
Department of Machine Learning
B.M.S. College of Engineering
Bangalore, India
mundrachetna@gmail.com

Deeksha Jayanth
Department of Machine Learning
B.M.S. College of Engineering
Bangalore, India
deekshajayanth02@gmail.com

Sonia Maria Dsouza
Department of Machine Learning
B.M.S. College of Engineering
Bangalore, India
s1985md@gmail.com

***Abstract*—This research paper presents an autonomous system that encompasses node detection, centroid calculation, distance measurement, shortest path planning, and command execution. The system is designed to detect nodes, calculate their centroids, and measure distances in the layout of the map represented as a graph. It also incorporates the capability to give autonomous commands such as "right", "left", "straight", "U-turn" "pick up," and "drop" when approaching a node. Moreover, the system prioritizes box pickup and drop-off based on colour-coded areas, including manufacturing, distribution, collection, and drop-off zones. By combining these functionalities, the system offers a comprehensive solution for automated navigation, box handling, and efficient path planning within complex environments, ultimately enhancing operational efficiency.**

***Keywords*—node detection, edge detection, Path planning, colour-coded box handling, navigation commands**

I. INTRODUCTION

In today's dynamic and interconnected world, efficient navigation and handling of objects within complex networks play a pivotal role in optimizing operational efficiency. This project focuses on the development of an autonomous system that encompasses node detection, track or edge detection, shortest path planning, and efficient colour-coded box handling within such networks.

The system leverages computer vision techniques to accurately detect nodes within a complex graph. By analysing the spatial distribution of nodes, centroid calculation enables better resource allocation and decision-making. Distance measurement algorithms further optimize path planning, facilitating the shortest and most efficient routes for box pickup and transportation.

Moreover, the system integrates autonomous command execution capabilities, enabling it to autonomously giving instructions such as turning right, left, moving straight, take U-turn picking up, and dropping off when approaching nodes. This reduces the reliance on manual intervention and minimizes human errors, enhancing operational efficiency.

Additionally, the system considers colour-coded areas, including manufacturing, distribution, collection, and drop-off zones. By prioritizing box pickup and drop-off based on colour and designated areas, the system streamlines operations and minimizes unnecessary movements.

The project aims to provide a comprehensive and efficient solution for automated navigation, box handling, and optimized path planning within complex networks. By implementing this system, operational efficiency can be significantly improved, leading to enhanced productivity and cost-effectiveness in various settings, such as manufacturing facilities, distribution centers, collection areas, and drop-off zones.

II. LITERATURE SURVEY

Y. Wan et al., [1] proposed "Edge Detection Algorithm Based on Grey System Theory Combined with Directed Graph" This research paper introduces a new take on edge detection algorithm for directed graph using grey system theory for profiled fibre contour extraction. It utilizes grey prediction and Niblack algorithm to obtain a strong edge and binary image. Edge linking and ROI calculations are performed, followed by flooding algorithm and contour tracking for outer contour extraction. In contrary to this we use simple and direct approach using thresholding techniques. It works sufficiently well with noise elimination and reduced false detection for the map provided thereby eliminating the complexity in the algorithm and higher system requirements.

Faris Adnan Padhilah et al., [2] proposed "Comparison Between Image Processing Methods for Detecting Object Like Circle". Procedure for circle detection with HSV and CHT (Circle Hough Transform) along with their combination is presented in this paper. The advantages and disadvantages of each method are discussed, including data processing speed, algorithm reliability, and limitations. The experiments involve a table tennis ball placed in different areas. This is an efficient circle or circular object detection algorithm. We propose a different approach by not treating the nodes on the graph as circles, instead we perceive the nodes as dense cluster of points of greater intensity than the rest of the graph. This approach allows us to obtain the nodes by applying a series thresholding functions and filters wherein we arrive at the densest area on the graph from which we obtain the coordinates of the center points approximately to the accuracy needed by a line following robot to locate the nodes and navigate seamlessly.

M. Aasim Qureshi et al., [7] proposed "An edge-wise linear shortest path algorithm for non-negative weighted undirected graphs." The paper proposes shortest path algorithm for unidirected-weighted (non-negative) graph. It focuses on finding an efficient path between a source and destination, rather than calculating the shortest path to all other nodes. It is unclear if the algorithm performs well on graphs with high node or edge densities, as these scenarios can pose challenges for efficient path finding algorithms. One potential disadvantage of this algorithm is that it may not perform optimally in scenarios where the road network exhibits non-hierarchical growth and situations where longer paths can have lower costs. This limitation could hinder its effectiveness for vehicle routing problems in certain real-world scenarios. Our approach does not require the graph to be hierarchal. It works correctly in any layout taking into consideration the nodes and weighted edge

III. METHODOLOGY

A. *Processing and feature extraction from the given map (graph)*

The image of the layout of the region is made into a graph along with nodes at every junction points. Three variations of the image of the map image are accepted as input. First variation is of the layout only along with nodes (Fig. 2), second variation is the layout along with the coloured boxes representing the different packages are placed in the respective positions (Fig. 7), the third variant (Fig. 8) is the layout along with the divisions of manufacturing unit (red), collection area (green), distribution area (blue) and drop off area (yellow) which is marked by assigning colours.

Primarily, feature detection has to be done in three different cases. Firstly, in the basic map where the nodes and the tracks are detected and the locations and distances are calculated precisely by applying a series of Open CV functions to deconstruct the given image and obtain the required details.

Basic layout processing:

The given basic map represented in (Fig. 2) shows the all-possible paths in the given region, junctions marked as nodes and straight and curved paths marked with black lines. By applying a list of Open CV functions such as cv2.cvtColor(), cv2.GaussianBlur(), cv2.dilate(), cv2.erode(), cv2.canny() in a specific order and fine tuning the parameters we deconstruct the images such that only the junction points are obtained as spots (Fig. 2). this image is further used to obtain the coordinates of the centre points (Fig. 4) of the nodes which we further use in navigation.

After obtaining the coordinates of the nodes in the original image, the nodes are masked completely, using the coordinate values leaving behind only the black lines representing tracks (Fig. 5). Using this image, the precise distances of all the tracks are calculated, and these distances are further assigned as the weight to the respective tracks (edges) (Fig. 6). After all the details obtained, the nodes and tracks (edges) are indexed and the nodes that form the end points of a particular track are mapped to that track and stored accordingly.

Processing the layout with coloured boxes and colour patches:

The second (Fig. 7) and third (Fig. 8) variants of the layout with coloured boxes and colour patches respectively are passed through a colour detection function to check the colour and its parity with the requirement of the command given to the bot. We have considered four colours namely, red, yellow, blue and green using which the boxes are colour coded (Fig. 7). The HSV value ranges of all the given colours of the boxes are computed.

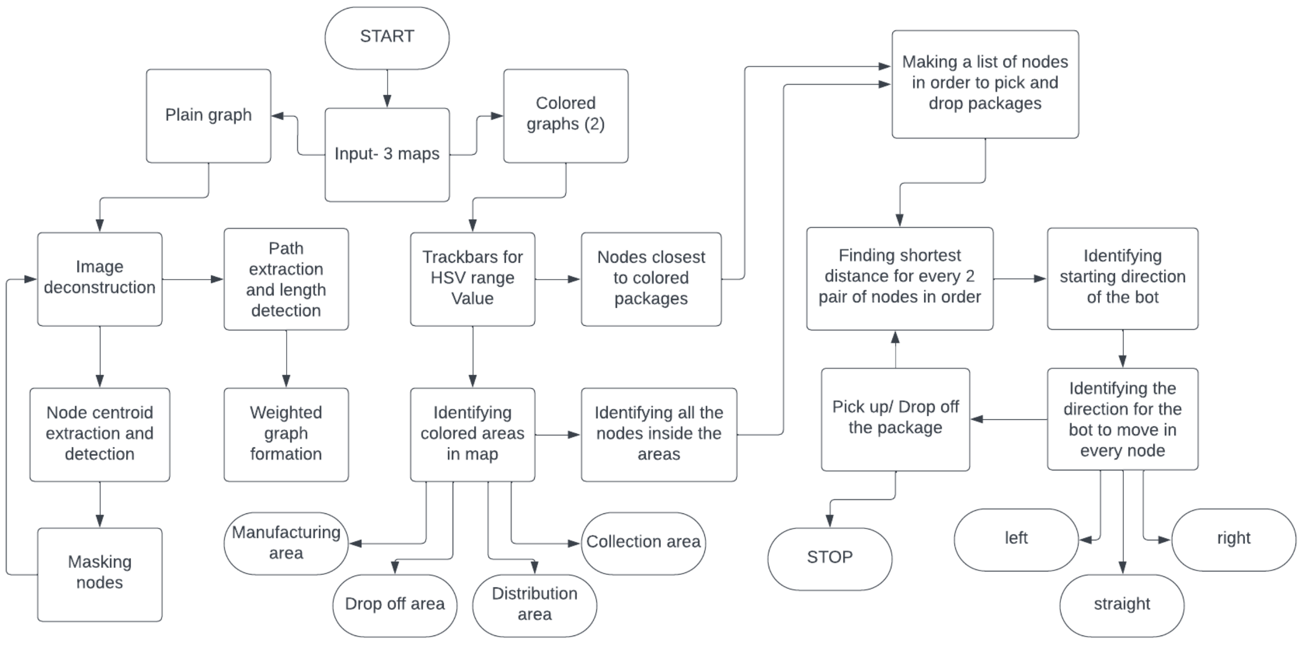
START
Input- 3 maps
Plain graph
Colored graphs (2)
Image deconstruction
Path extraction and length detection
Node centroid extraction and detection
Weighted graph formation
Masking nodes
Trackbars for HSV range Value
Nodes closest to colored packages
Identifying colored areas in map
Identifying all the nodes inside the areas
Manufacturing area
Drop off area
Distribution area
Collection area
Making a list of nodes in order to pick and drop packages
Finding shortest distance for every 2 pair of nodes in order
Identifying starting direction of the bot
Identifying the direction for the bot to move in every node
Pick up/ Drop off the package
STOP
left
straight
right

Fig. 1. Flowchart

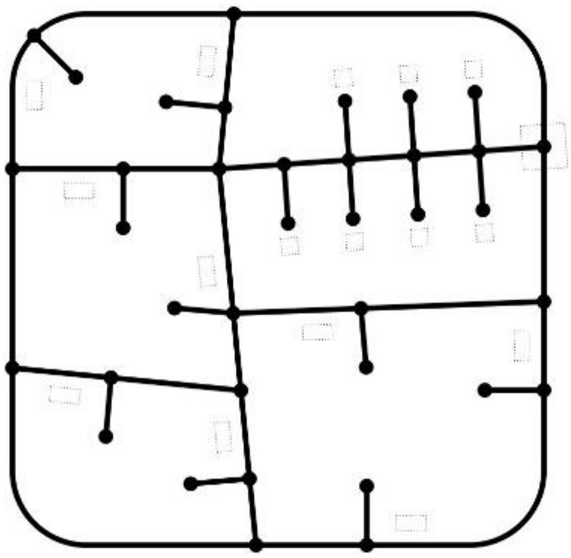
Fig. 2. Basic Layout *(Original Map)*

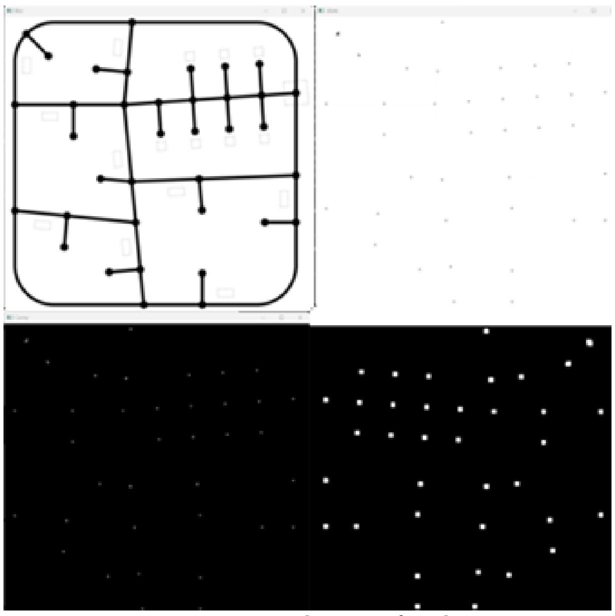
Fig. 3. Incremental process of Node extraction

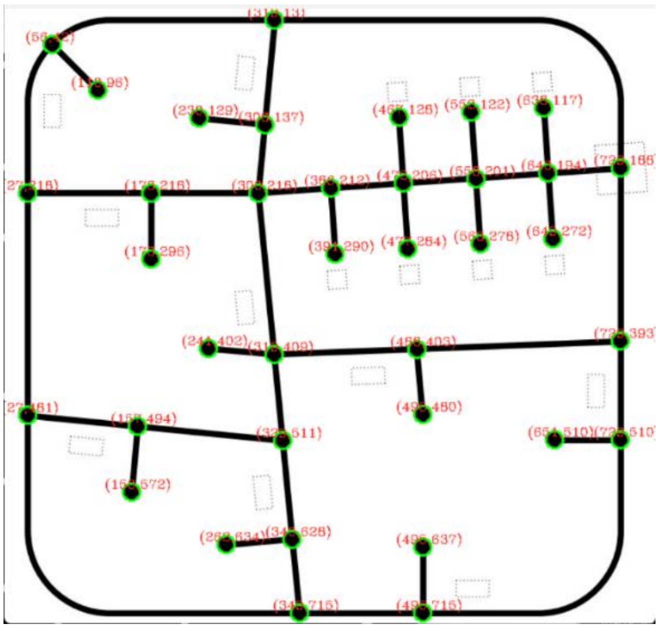
Fig. 4. Coordinates of centroids of the Nodes

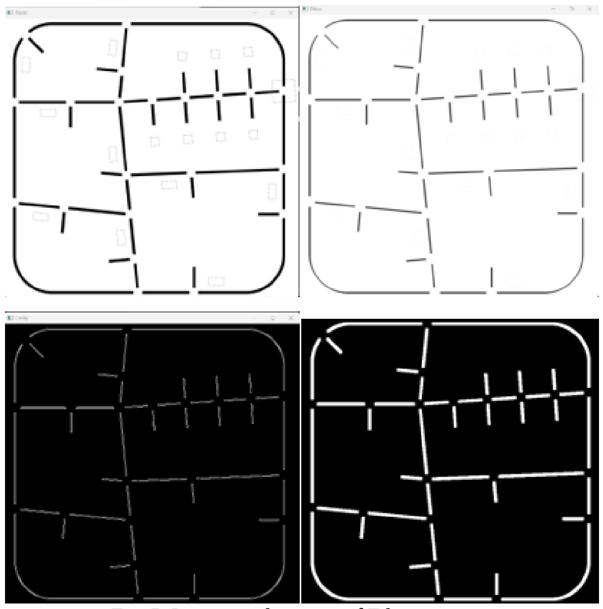
Fig. 5. Incremental process of Edge extraction

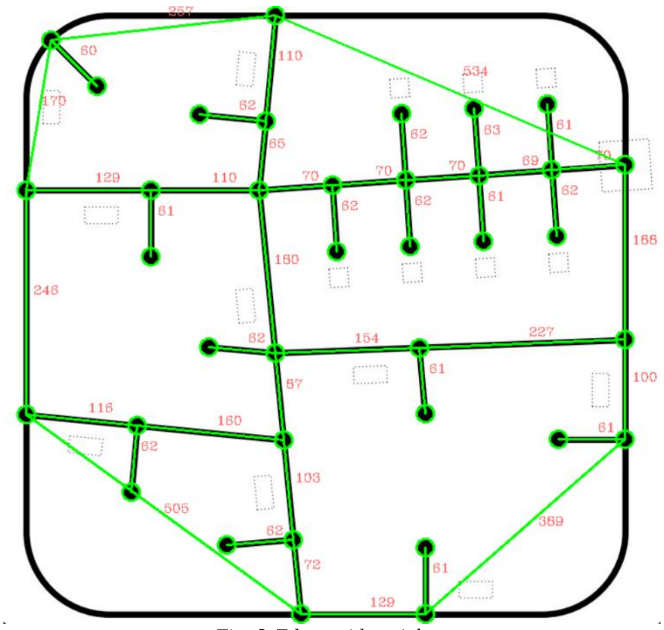
Fig. 6. Edges with weights

Trackbars for H-Hue, S-Saturation and V-Value (Brightness) are created and the feasible range of values are obtained. By using the obtained values, the colours of the boxes are determined and the boxes are categorized accordingly.

The manufacturing area, distribution area, collection area, and the drop off area are also colour coded (Fig. 8), according to which the areas are identified. The colours are identified here using the same method mentioned above, using range of HSV values.

All the above-mentioned processing of the image and functions are applied as and when required autonomously. The program incorporates image processing, detecting the nodes and lanes (tracks or edges) and extracting the necessary formation form the three images provided as input and autonomously performs all the functions and generates commands to be given to the bot to navigate without any human interference.

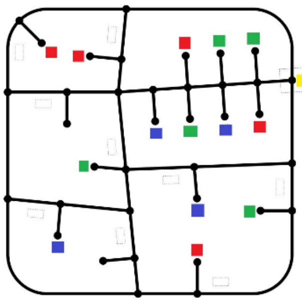
Fig. 7. Location of boxes

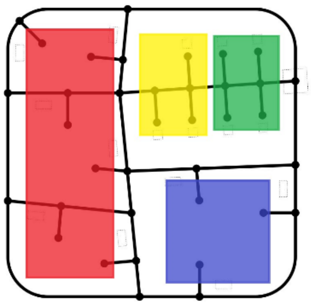
Fig. 8. Coloured Areas

Libraries and functions used:

OpenCV-Python:

OpenCV is a powerful open-source computer vision library used in research papers for video and image processing tasks, object detection, tracking, and more. With OpenCV-Python, you can perform tasks such as reading and writing images and videos, applying various image processing operations (like filtering, thresholding, morphological operations, etc.), performing geometric transformations (such as scaling, rotating, and warping), detecting and recognizing objects in images, and much more.

Major functions used in OpenCV:

cv2.findContours(): This function is used to detect and extract contours from binary images. The function takes an input binary image and returns a list of contours along with their hierarchy. Mathematically, the cv2.findContours() function implements the concept of contour detection using the concept of level sets. finding the level sets at different intensity thresholds, the function identifies the contours present in the image.

cv2.moments(): This function calculates various moments of a contour or binary image, providing information about its shape. Mathematically, moments are calculated using integrals and are represented by scalar values. The centroid (cx, cy) is obtained as cx = m10/m00 and cy = m01/m00, where mij is the central moment of order (i+j). m00 is the zeroth order moment, which represents the total sum of pixel intensities and is used to calculate the object's area. m10 and m01 are the first-order moments m01 and m10 corresponds to the sum of all non-zero pixels in x-coordinate and y-coordinate respectively. I(x, y) represents the intensity or pixel value at coordinates (x, y). $\overline{x}$ and $\overline{y}$ represent the coordinates of the centroid.

$$m00 = \sum_x \sum_y I(x,y)$$

$$m10 = \sum_x \sum_y x\, I(x,y)$$

$$m01 = \sum_x \sum_y y\, I(x,y)$$

$$\overline{x} = \frac{m01}{m00} = \frac{1}{A}\int_a^b x f(x)dx$$

$$\overline{y} = \frac{m10}{m00} = \frac{1}{A}\int_c^d y f(y)dy$$

cv2.GaussianBlur(): This function is used for blurring and reducing noise in the image using Gaussian smoothing. The equation for a 2D Gaussian kernel with standard deviation σ is:

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2-y^2}{2\sigma^2}}$$

This function computes the weighted mean of pixel values within an image by employing a Gaussian kernel. The degree of blurring is regulated by the standard deviation (σ) parameter, where larger values result in more significant blurring. This smoothing operation is useful for noise reduction and pre-processing steps in image analysis and computer vision tasks.

cv2.dilate(): This function is used to perform dilation on binary images or masks. Dilation is a morphological operation that expands the boundaries of foreground regions. Mathematically, dilation is performed by convolving the image with a structuring element. The equation for dilation at a particular pixel (x, y) is, output(x, y) = max of pixels in the neighbourhood defined by the structuring element. The cv2.dilate() function replaces each pixel with the maximum value within its neighbourhood, expanding the shape and size of foreground regions. It is commonly used in tasks like object detection, noise removal, and image segmentation.

cv2.canny(): This function is used for edge detection in images. It applies the Canny edge detection algorithm, which involves various mathematical operations to find and highlight edges in an image. The cv2.Canny() function implements these steps, taking the input image and the threshold values as parameters. It returns a binary image highlighting the detected edges.

cv2.inRange(): This function is used to create a binary mask by thresholding pixel values within a specified range. Mathematically, the function checks if each pixel value in the input image lies within the specified range and assigns a binary value (0 or 255) accordingly.

cv2.morphologyEx(): This function is used for advanced morphological operations on images, such as erosion, dilation, opening, closing, and more. These operations transform the appearance and arrangement of objects within an image. Morphological operations involve the convolution of the image with a structuring element. The cv2.morphologyEx() function applies these operations, taking the input image and a structuring element as parameters. It enables tasks like noise removal, shape analysis, and object extraction in image processing and computer vision applications.

NumPy - Functions used: np.ones(), np.unit8, np.zeros_like(), np.array()

Maths - Functions used: math.degrees(), math.atan(), math.acos(), math.dist()

B. Path Planning and Navigation

After creating a dictionary that includes all the units and their respective nodes along with package colours, a perfect plan needs to be developed with the least number of jumps between nodes. For example, if there are 3 packages in the manufacturing unit to be dropped off at the drop-off area and

4 packages in the collection unit to be dropped off at the distribution area, it is crucial to decide the order in which the packages should be picked up and dropped off, minimizing the distance travelled by the bot.

Since the starting node of the bot is already determined, we need to calculate the distance between the starting node and the nodes that belong to the manufacturing unit or collection unit. The node with the shortest distance from the starting position becomes the bot's destination. Once the bot reaches the destination, the second node becomes the source node, and it’s destination, either the distribution area or the drop-off area, is already determined. The bot then picks up and drops off the corresponding package. This process continues with the third node becoming the source node, and the next package is determined using the same method as before. Each time a node is encountered, it is removed from the dictionary. This process continues until there are no nodes left in the dictionary.

Here is a Sample dictionary - {'manufacturing unit': {3: 'blue', 5: 'blue', 16: 'green', 30: 'red'}, 'drop off': {17: 'green', 18: 'red', 31: 'blue'}, 'collection area': {19: 'green', 20: 'blue', 32: 'blue', 33: 'red'}, 'distribution area': {2: 'blue', 8: 'red', 11: 'green'}}

From Fig. 7, taking yellow node as the starting node, second node to be travelled by the bot would be green node of manufacturing unit, since it is the closest to start. Taking above example, package is dropped off to respective node in distribution area, and from the node there, it travels to blue node of manufacturing area to pick up the next package. This process continues until all packages are dropped off. The distance between two nodes when determining the next pick up package is calculated using distance formula.

The distance formula to find the distance between $A_1(x_1,y_1)$ and $A_2(x_2, y_2)$ is

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

The shortest path between every two nodes can be calculated efficiently using Floyd Warshall algorithm or any shortest planning algorithm for weighted graphs. List data structure is used to store the nodes in the order they need to be visited.

Navigation

Once the list of nodes is ready, instructions are given to the bot each time it encounters a node, indicating whether to pick up a package, drop off a package, start, stop, go left, right, or straight. The maths used for this involves calculating the sine of the angles between two lines. The initial direction of the bot is determined based on its next coordinates. After that, the sine of the angle between three points, let's say $A_1(x_0,y_0)$, $A_2(x_1,y_1)$, $A_3(x_2,y_2)$ is calculated. If the angle is close to 180°, the sine value will be close to 0, indicating that the bot should go straight. If the sine value is close to 1 or -1, it means the bot should go left or right, respectively. If the sine value of the angle is not close to 0, 1, or -1, it means the line is a curved line.

The different scenarios to occur are in Fig. 9 which illustrates points A, B, and C. To decide whether the bot should go left or right, assuming the minimum distance between every line is at least d, the conditions in Table I apply to determine the bot’s direction.

To identify the curved edges, α should be an acute angle or an obtuse angle. In the case of curved edges, we check if the slope of the line is exactly 1 or not. If the slope of the second line is 1, it indicates that it is the line starting from the curved edge, as shown in Fig. 11 The directions for this line can be obtained from Table I. This approach works efficiently because the tangent to the curve also has a slope of 1. Therefore, the line 34-35 (Fig. 11) is perfectly perpendicular to the curve.

TABLE I.

	x_1-x_2>d ***(left)***	**x_2-x_1>d** **(right)**	**y_1-y_2>d** ***(up)***	**y_2-y_1>d** ***(down)***
x_0-x_1>d ***(left)***	-	-	Sin(α) ≅1 right	Sin(α) ≅-1 left
x_1-x_0>d ***(right)***	-	-	Sin(α) ≅1 left	Sin(α) ≅-1 right
y_0-y_1>d ***(up)***	Sin(α) ≅ -1 left	Sin(α) ≅1 right	-	-
y_1-y_0>d ***(down)***	Sin(α) ≅-1 right	Sin(α) ≅1 left	-	-

The row shows the direction of bot it is currently in, while the column shows the direction of bot in which it has to move forward to.

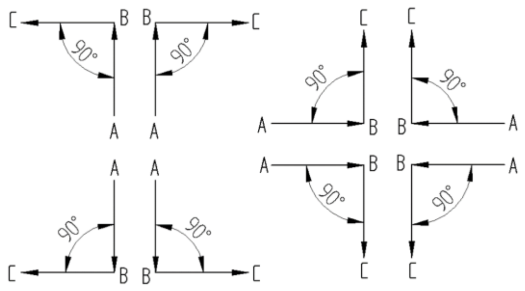

Fig. 9. All possible directions

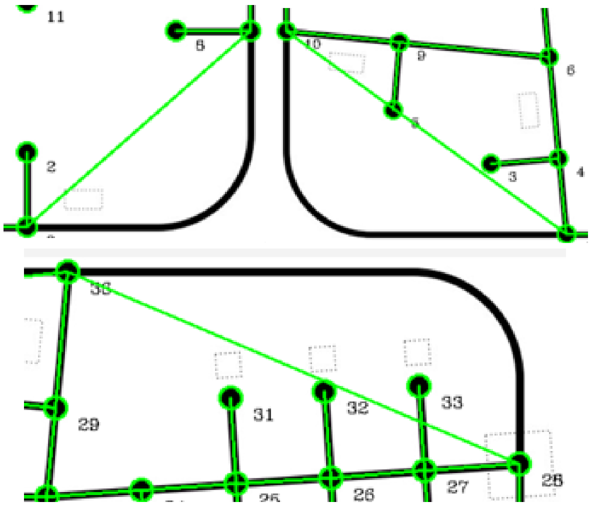

Fig. 10. Curved edges

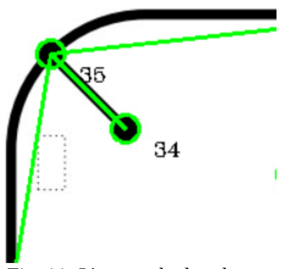

Fig. 11. *Line* attached to the curve

In Fig. 10 the angles between the lines (4,1,10), (8,0,2) and (36, 28, 27) all form acute angles with each other.

In case the slope of the second line is not 1, then the bot has to continue moving straight, and the initial position of the bot needs to be adjusted. Once the initial direction of the bot is changed, we can refer to Table I for navigation instructions.

Functions used to find the sin of angles and slope of lines are as follows:

Math.asin() - This method calculates and returns the arcsine (inverse sine) of a given number.

Math.deg() - This function is used for converting an angle from radians to degrees.

Math.atan() – This method provides the arctangent (inverse tangent) of a number 'x' as a numeric value within the range of -π/2 to π/2 radians.

Slope formula –

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

where (x_1,y_1), (x_2,y_2) are two coordinates on the line, and inclination of one line with respect to other is given as -

$$\tan \emptyset = \frac{m_1 - m_2}{1 - m_1 m_2}$$

IV. RESULTS

The algorithm takes the maps as the input and generates a sequence of commands which includes : 'S' – Straight, 'R' – Right, 'L' – Left, 'U' – U-turn, 'P' – Pick-up, 'D' – Drop, 'O' – Stop. The list of commands obtained is relayed to the robot through Bluetooth, so whenever a node is encountered, the bot will know which command to perform, leading it to the right destination. The output of the given problem –

```
['S', 'S', 'S', 'S', 'S', 'R', 'L', 'P', 'R', 'L', 'S',
'R', 'D', 'R', 'S', 'L', 'P', 'L', 'R', 'S', 'R', 'D',
'L', 'S', 'L', 'S', 'S', 'S', 'S', 'S', 'L', 'P', 'R',
'S', 'R', 'D', 'R', 'S', 'R', 'P', 'L', 'S', 'S', 'L',
'L', 'R', 'D', 'L', 'L', 'S', 'R', 'P', 'L', 'S', 'S',
'R', 'S', 'L', 'D', 'R', 'S', 'L', 'S', 'R', 'L', 'P',
'R', 'L', 'S', 'R', 'S', 'L', 'D', 'L', 'S', 'R', 'P',
'R', 'R', 'S', 'S', 'R', 'D', 'R', 'R', 'S', 'S', 'S',
'R', 'S', 'S', 'L', 'P', 'R', 'S', 'S', 'L', 'S', 'S',
'S', 'L', 'L', 'D', 'L', 'S', 'S', 'O']
```

Fig. 12. Output showing command sequence to be followed by the bot.

V.CONCLUSION

In conclusion, the proposed autonomous system offers significant advantages in various operational settings. It enables accurate node detection, centroid calculation, edge detection, distance measurement, shortest path planning, autonomous command generation. The system's consideration of colour-coded areas optimizes box handling, while its integration of computer vision techniques enhances navigation efficiency. By streamlining operations, reducing human error, and prioritizing tasks, the system improves operational efficiency in manufacturing, distribution, collection, and drop-off zones. Overall, this comprehensive solution for automated navigation and box handling has the potential to significantly enhance operational efficiency across diverse settings.

VI. FUTURE SCOPE

The future scope of the proposed system lies in further advancements and enhancements to its capabilities. Potential avenues for development include incorporating machine learning algorithms to improve node detection accuracy and tracking robustness. Additionally, integrating real-time sensor data and advanced mapping techniques could enhance the system's ability to adapt to dynamic environments. Furthermore, exploring the integration of robotic arms or automated material handling systems could enable seamless box pickup and drop-off operations. Moreover, the system could be extended to support multi-robot coordination and collaboration, allowing for increased efficiency and scalability in large-scale operations. Overall, future developments could unlock even greater automation, efficiency, and adaptability in navigating and handling tasks within complex networks.

REFERENCES

[1] Jingkuan Song, Zhilong Zhou, Lianli Gao, Xing Xu, and Heng Tao Shen. 2018. Cumulative Nets for Edge Detection. In Proceedings of the 26th ACM international conference on Multimedia (MM '18). Association for Computing Machinery, New York, NY, USA, 1847–1855.

[2] Y. Wan, S. Jia and D. Wang, "Edge Detection Algorithm Based on Grey System Theory Combined with Directed Graph," 2013 Seventh International Conference on Image and Graphics, Qingdao, China, 2013, pp. 180-185, doi: 10.1109/ICIG.2013.42.

[3] Hanghang Tong, Christos Faloutsos, and Yehuda Koren. 2007. Fast direction-aware proximity for graph mining. In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '07). Association for Computing Machinery, New York, NY, USA, 747–756.

[4] Faris Adnan Padhilah and Wahidin Wahab. 2018. Comparison Between Image Processing Methods for Detecting Object Like Circle. In Proceedings of the 2018 International Conference on Digital Medicine and Image Processing (DMIP '18). Association for Computing Machinery, New York, NY, USA, 33–36.

[5] Sabine Storandt. 2018. Sensible edge weight rounding for realistic path planning. In Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '18). Association for Computing Machinery, New York, NY, USA, 89–98.

[6] J. Sellen, "Direction weighted shortest path planning," Proceedings of 1995 IEEE International Conference on Robotics and Automation, Nagoya, Japan, 1995, pp. 1970-1975 vol.2, doi: 10.1109/ROBOT.1995.525552.

[7] M. Aasim Qureshi, Mohd Fadzil Hassan, Sohail Safdar, Rehan Akbar, and Rabia Sammi. 2009. An edge-wise linear shortest path algorithm for non negative weighted undirected graphs. In Proceedings of the 7th International Conference on Frontiers of Information Technology (FIT '09). Association for Computing Machinery, New York, NY, USA, Article 67, 1–4.