www.ThePharmaJournal.com

The Pharma Innovation

ISSN (E): 2277-7695
ISSN (P): 2349-8242
NAAS Rating: 5.23
TPI 2023; 12(12): 3398-3407
© 2023 TPI
www.thepharmajournal.com
Received: 13-09-2023
Accepted: 29-10-2023

Dr. Chirag S Matholiya
Ph.D., Department of Farm Machinery and Power Engineering CAET, Anand Agricultural University, Godhara, Gujarat, India

Dr. Piyush R Balas
Senior Technical Assistant, Department of Farm Machinery and Power Engineering CAET, Junagadh Agricultural University, Junagadh, Gujarat, India

Sanjaykumar J Pargi
Senior Research Assistant, CAET, Anand Agricultural University, Godhara, Gujarat, India

Dr. Vishal V Agravat
Ph.D., Department of Farm Machinery and Power Engineering CAET, Anand Agricultural University, Godhara, Gujarat, India

Dr. Pankaj Gupta
Professor and Head, Department of Farm Machinery and Power Engineering CAET, Anand Agricultural University, Godhara, Gujarat, India

Corresponding Author:
Dr. Chirag S Matholiya
Ph.D., Department of Farm Machinery and Power Engineering CAET, Anand Agricultural University, Godhara, Gujarat, India

Development of an algorithm for crop row detection for autonomous fertilizer side dressing machine

Dr. Chirag S Matholiya, Dr. Piyush R Balas, Sanjaykumar J Pargi, Dr. Vishal V Agravat and Dr. Pankaj Gupta

Abstract
Agricultural crop production involves many operations, out of which fertilizer application is one of the important and least exploited operations. At present fertilizer, especially in split doses applied manually. But, labour scarcity is becoming a problem day by day. To reduce the labour problem and to make agriculture lucrative, the use of modern technologies such as robotics, automation, etc. is the need of the hour. Considering all these, an autonomous machine was developed for fertilizer side dressing application. In the fertilizer side dressing technique, the exact metered quantity of fertilizer was applied near the periphery of the plant which overcome certain problems.

An autonomous machine was developed using vision-based and sensor-based technology. In vision-based technology, the image processing technique was used to identify the row crop and allowed the machine to run in between the rows by correcting the lateral error. The principle of the machine was to run the machine in between the row crop and drop the exact metered quantity of fertilizer near the plant periphery.

In the field test, machine trajectory parameters like tracking lateral error, angle of deviation and reaction time were found significant at 5% level of significance at camera height of 60 cm and camera angle of 45° in cotton and okra crops. The minimum tracking lateral error, angle of deviation and reaction time were found to be 4.37 cm, 9.4° and 0.47 s for cotton crop and 3.35 cm, 7.01° and 0.38 s for okra crop, respectively at camera height of 60 cm and camera angle of 45°.

Keywords: Autonomous, side dressing, tracking lateral error, angle of deviation, reaction time

Introduction
This research paper deals with the procedure of crop row detection algorithm which is used to achieve the objectives of the research problem which is the development and performance evaluation of an autonomous fertilizer side dressing machine.

Development of an Algorithm for Path Detection
For the machine guidance system, the image processing technique was found suitable according to the review analysis. OpenCV software was used for generating the code for image processing.

OpenCV
OpenCV (Open-Source Computer Vision Library) is an open-source library for computer vision, machine learning, and image processing techniques. OpenCV is a cross-platform library using which we can develop real-time computer vision applications. It mainly focuses on image processing, video capture and analysis including features like face, line and object detection. Open-source computer vision is a library of programming functions mainly aimed at real-time computer vision, originally developed by Intel and it was later supported by Willow Garage. OpenCV is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. OpenCV supports numerous programming languages like Python, C++, Java, etc. OpenCV has a modular structure, which means that the package includes several shared or static libraries. The following modules are available:

- Core functionality (core) - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.
- Image Processing (imgproc) - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.
- Video Analysis (video) - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.
- Camera Calibration and 3D Reconstruction (calib3d) - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
- 2D Features Framework (features2d) - salient feature detectors, descriptors, and descriptor matchers.
- Object Detection (objdetect) - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).
- Video I/O (videoio) - an easy-to-use interface to video capturing and video codecs

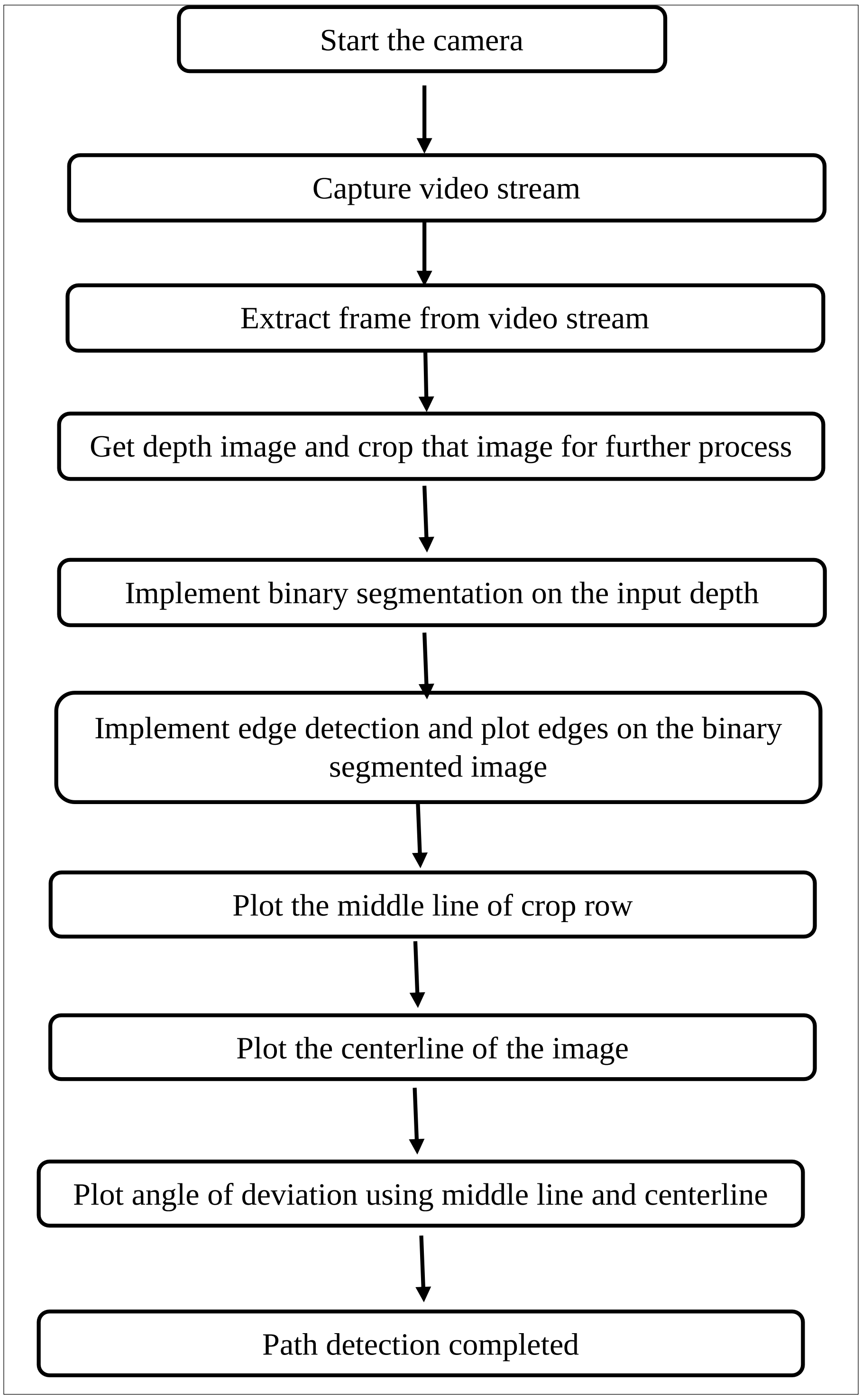
Python
Python is a popular and widely used general-purpose and high-level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code. Python aides and supports various programming ideal models, moreover, as question arranged, basic and handy programming or procedural styles. It has colossal features of dynamic sort framework and customized memory administration and contains a goliath and standard library. NumPy and pprint libraries are available in Python which was used in the algorithm development. NumPy is a library consisting of multidimensional array objects and a collection of routines for processing of array. Using NumPy, mathematical and logical operations on arrays can be performed. pprint was used to print in array format in the console. Python 3 was used to develop an algorithm for path detection within the crop area.

Materials and Methods
Image processing is the technique that deals with processing digital images through an algorithm. Digital images from the video stream get processed into the microcontroller. Experiments were conducted in the field with three independent parameters namely camera height (3 levels), camera angle (3 levels) and crops (cotton and okra) (Table 1). Machine movement was identified based on the result of Tracking lateral error, Angle of deviation, and Reaction time.

Table 1: Independent and Dependent Parameters

Sr. No.	Variables	Parameters	Levels
1	Independent Parameters	Camera height (cm)	$H_1 = 40$ $H_2 = 60$ $H_3 = 80$
		Camera angle (degree)	$\theta_1 = 30$ $\theta_2 = 45$ $\theta_3 = 60$
		Crop	Cotton Okra
2	Dependent Parameters	Tracking lateral error, cm Angle of deviation, ° Reaction time, s	

Steps for Algorithm Development for Crop Row Detection
Several steps for path detection have been followed in image processing are mentioned in Flow 1.


Start the camera
Capture video stream
Extract frame from video stream
Get depth image and crop that image for further process
Implement binary segmentation on the input depth
Implement edge detection and plot edges on the binary segmented image
Plot the middle line of crop row
Plot the centerline of the image
Plot angle of deviation using middle line and centerline
Path detection completed

Flow 1: Flowchart for path detection

Step-1: Get input video stream
Two webcams were used to capture the video stream from the crop row. From these webcams, two video streams were captured.

```
import cv2
import numpy as np
from pprint import pprint
vid1 = cv.VideoCapture(1) # Camera Left
vid2 = cv.VideoCapture(0) # Camera Right
```

Here cv2 was used for accessing the camera of the system. Numpy was used for the numerical calculation of the image. print was used for printing data in the console (Only for the development debugging process). vid1 and vid2 were a variable that provides a live video stream of the camera.

Step-2: Extraction frame from video stream
One frame each was extracted from video stream of left and right camaras. The extracted frame was considered as left and right image. The extracted image from video stream is shown in Fig. 1.

```
while(True):
ret1, frame1 = vid1.read()
ret2, frame2 = vid2.read()
cv.imwrite(‘frame1.jpg’, frame1)
cv.imwrite(‘frame2.jpg’, frame2)
```

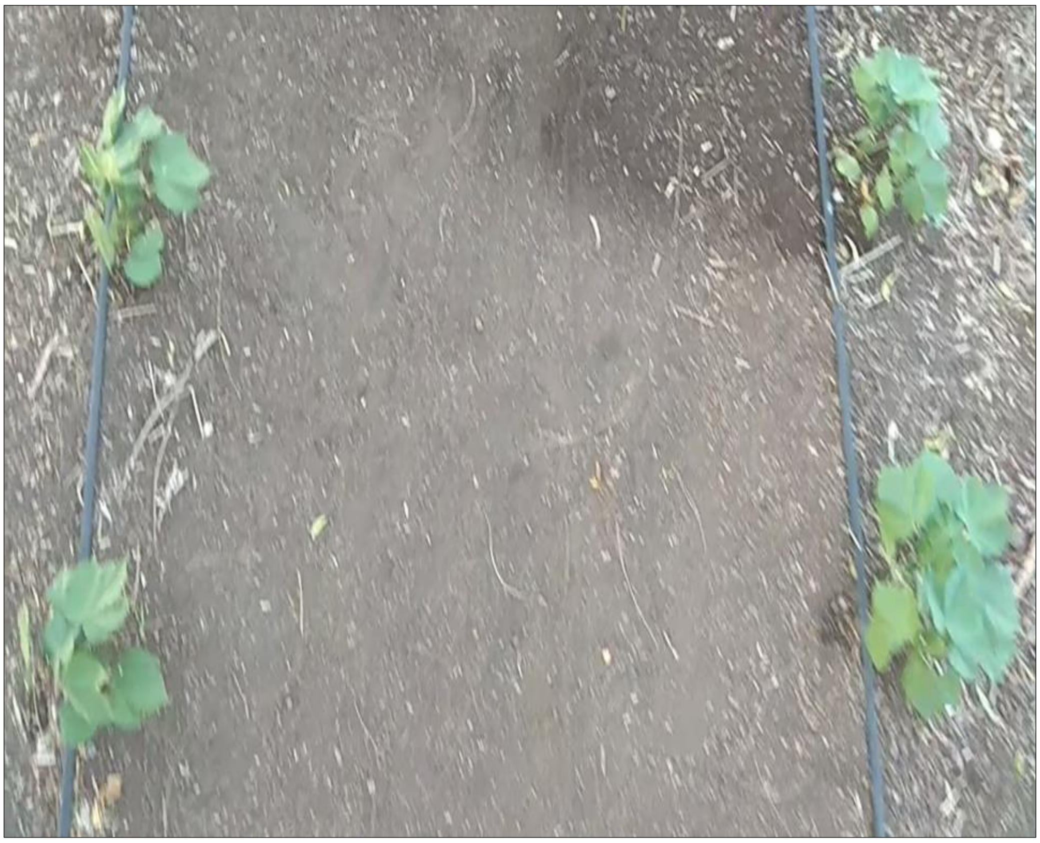


Fig 1: Frame extracted from video stream

Here while (True) command was used to constantly extract a frame from a video stream. ret (ret1, ret2) was a video frame ret which represents the FPS of the camera. The frame (frame1, frame2) was a single image data extracted from a video.

Step-3: Get desired size depth image

Left and right images were merged and get a merged image as depth image.

```
imgL = cv.imread('frame1.jpg',0)
imgR = cv.imread('frame2.jpg',0)
stereo = cv. StereoBM create (num Disparities = 16, blockSize=15)
disparity = stereo.compute(imgL,imgR)
cv.imwrite('depth.jpg', disparity)

# Image cropping
img = cv2.imread('depth.png')
height, width, channels = img.shape
img = img[0:height-int(height/4),int(width/5):width-int(width/5)]
cv2.imshow('croped',img)
```

Frames were stored as images. The data of frames were stored in imgL and imgR variables to access numerical data of the image. The depth image was got from imgL and imgR using StereoBM_create and stored as a depth image. The merged image was stored with the name "depth.jpg". The height, width and channel were stored and cropped according to the suitability of the image processing. The image was cropped by 25% of the height (height/4) because the remaining part of the height was not required for image processing. The image was cropped by 20% of the width (width/5) from both sides of the image because the image was seemed to be centered from the crop for future processes.

Binocular disparity refers to the difference in image location of an object seen by the left and right eyes resulting from the eyes' horizontal separation (parallax). The brain uses binocular disparity to extract depth information from the two-dimensional retinal images in stereopsis (Doxygen, 2021) [1-2]. The disparity is referred to the identification of the match point in two images. The disparity needs to be set in the code for getting depth image. The measurement of the disparity of the cameras is shown in Fig. 2. In our case, the disparity was found to be 16 mm with the help of using formula 1.

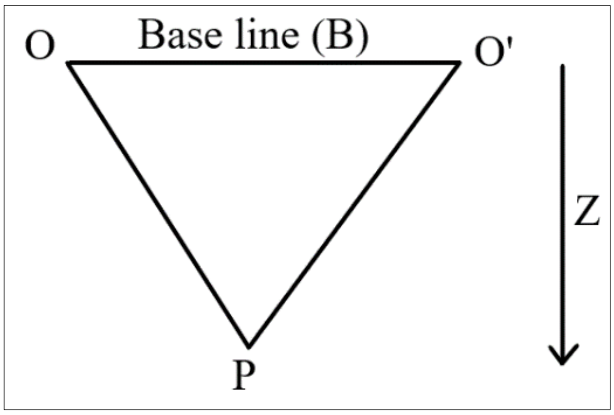


Fig 2: Disparity measurement of cameras

$$\text{Disparity} = \frac{BF}{Z} \qquad (1)$$

Where,

B = Distance between two cameras (23 cm)

F = Focal length of camera (Focal length = 41.55 mm)
Z = Distance from the camera to the ground (60 cm)

$$\text{Disparity} = \frac{23*41.55}{60}$$

Disparity = 16 mm

Step-4: Implement binary segmentation on depth image
Depth image was converted into grayscale and then binary segmentation was implemented using cv2.threshold function. The binary segmented image data were stored in the variable named binary Segmented.

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, binarySegmented =cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
```

A basic technique for object segmentation was called thresholding. Thresholding is a very popular segmentation technique, which was used for separating an object from its background. The thresholding process involved comparing each pixel value of the image (pixel intensity) to a definite threshold value. All the pixels of the input image were separated into 2 sets (Fari, 2013).

1. Pixels having intensity value lesser than the threshold.
2. Pixels having an intensity value larger than the threshold.

$$\text{Background} = \begin{cases} \text{maxVal} & \text{if pixel value} > \text{threshold value} \\ 0 & \text{otherwise} \end{cases}$$

Binary segmentation was done based on values 0 and 1. 0 value denoted to black pixel and 1 value denoted to white pixel. The threshold value range was 0 to 255. If the observed pixel value was found above the average pixel value, then it was depicted in maxVal and denoted to 1. This meant that white background was found. If the observed pixel value was found below the average pixel value, then it was denoted to 0. This meant that black background was found. The binary segmentation in image is shown in Fig. 3.

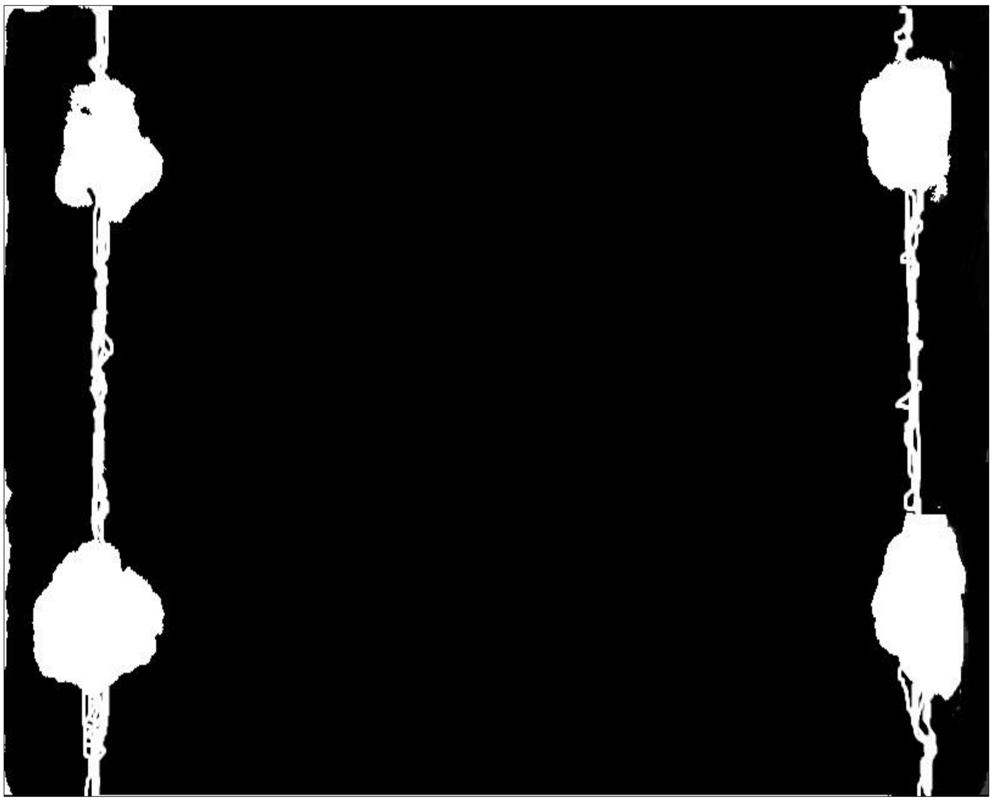


Fig 3: Binary segmented image

Step-5: Implement edge detection on the binary segmented image
Canny edge detection is a widespread edge detection technique. It was developed by John F. Canny. Edge detection technique is used to classify points in a digital image with discontinuities or sharp changes in the image brightness. The points which fall into image brightness and vary sharply are called the edges (or boundaries) of the image (Doxygen, 2021) [1-2].

```
edges = cv2.Canny(binarySegmented,100,200)
```

Two edges were formed in the image followed by two crop rows. edges data of images were stored in variable named edges. The edges of the crop row are indicated by a white continuous line in Fig. 4.

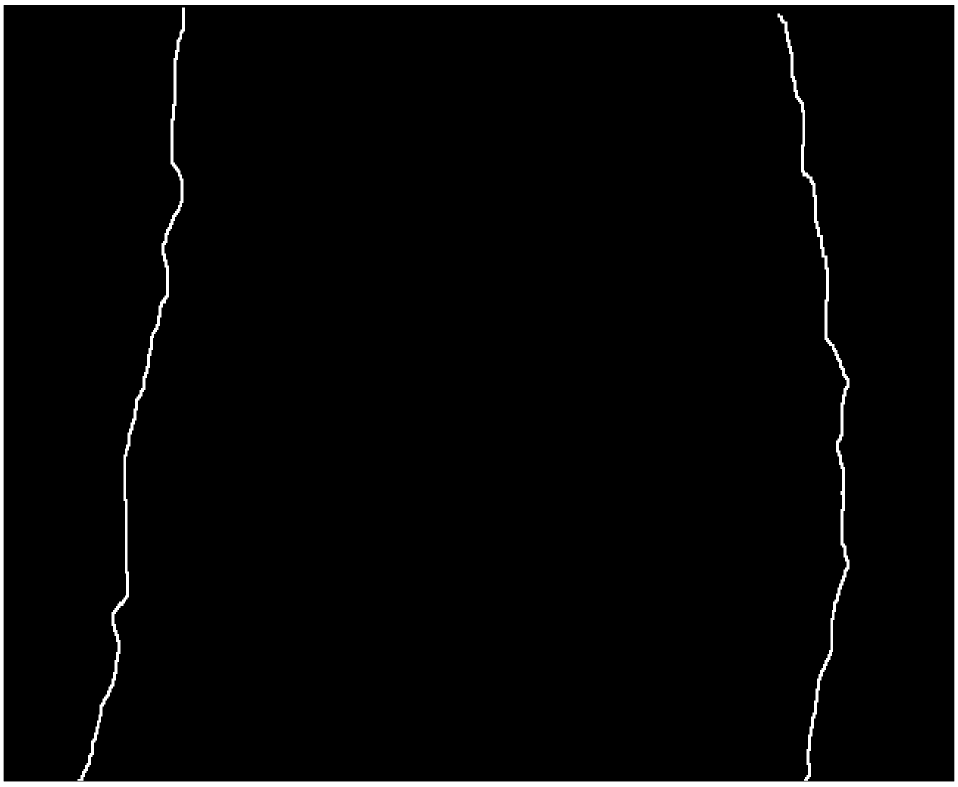


Fig 4: Canny edge detection in image

Canny edge detection

The smoothened image was filtered with a Sobel kernel in both horizontal and vertical directions to get the first derivative in the horizontal direction (Gx) and vertical direction (Gy). Edge gradient and direction for each pixel are mentioned in formula 2.

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \qquad (2)$$

Gradient direction is always perpendicular to edges and rounded to one of four angles representing vertical, horizontal and two diagonal directions. After getting gradient magnitude and direction, a full scan of the image was done to remove any unwanted pixels, which may not constitute the edge. For edge detection, every pixel was checked if it was a local maximum in its neighbourhood in the direction of the gradient.

In Fig. 5, point A was considered an edge in the vertical direction. The gradient direction was ordinary to the edge. Points B and C were in gradient directions. So, point A was checked concerning points B and C. If it formed a local maximum then it was considered for the next stage, otherwise it was suppressed (put to zero). In short, a binary image with "thin edges" was found as a result.

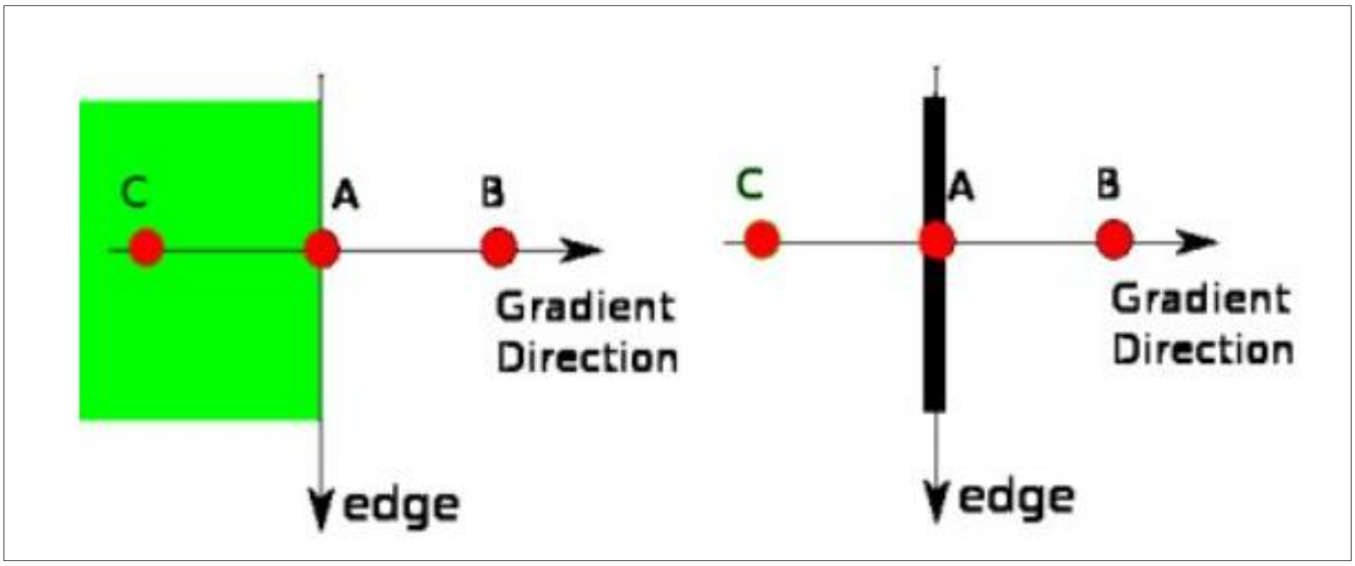


Fig 5: Gradient direction

Now, the number of edges was formed and the upcoming stage was decided which edges were edges or not. Threshold values need to be differentiated into two edges which are minVal and maxVal. The pixels having an intensity gradient more than maxVal are sure to edge and those pixels' intensity below minVal are sure to be non-edges. The pixels that lie in between two threshold values were classified as edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered as part of edges. Otherwise, they were discarded. Canny edge detection discontinuity is shown in Fig. 6.

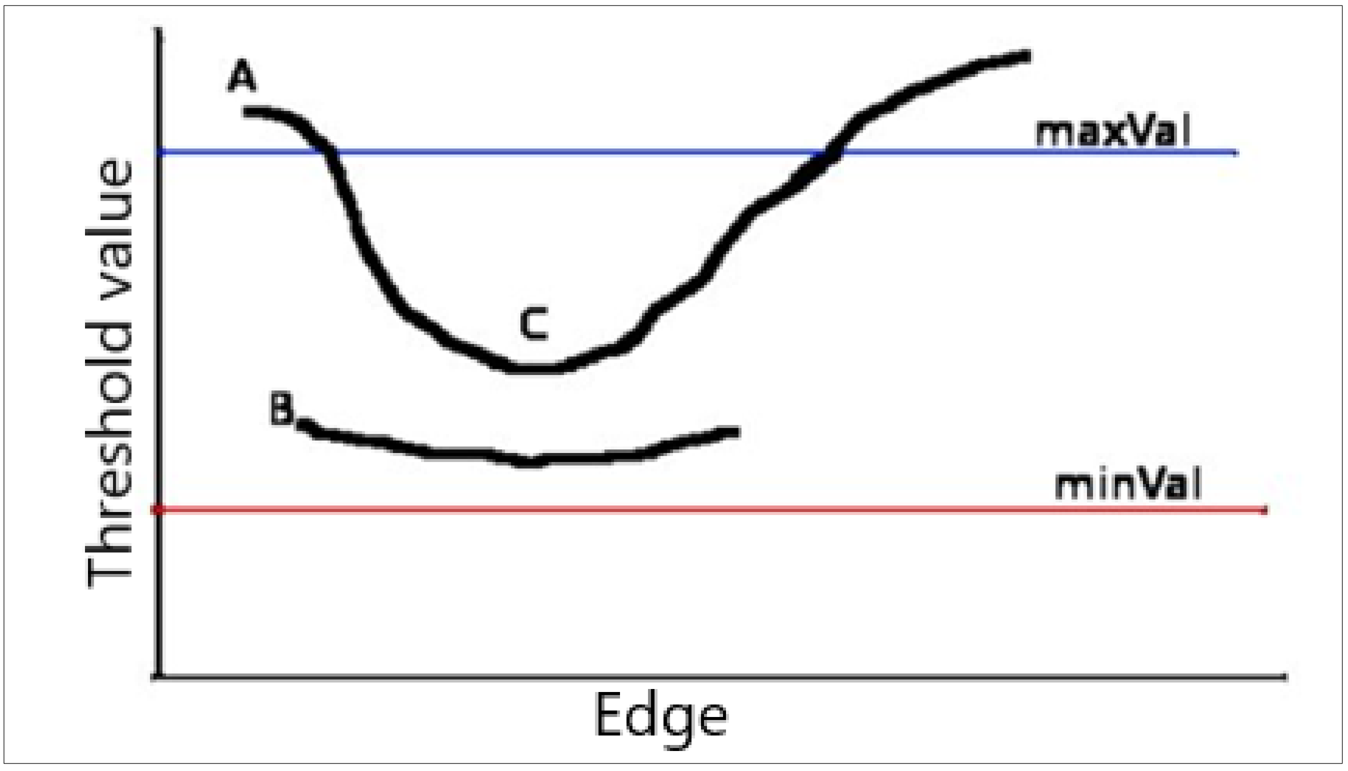


Fig 6: Canny edge detection discontinuity

In Fig. 6, edge A is above the maxVal, so it was considered as "sure-edge". Although edge C is below maxVal, it was connected to edge A, so it was considered a valid edge and got a full curve. But edge B is above minVal but it was not connected to any "sure-edge" so it was discarded. So, the minVal and maxVal were decided accordingly to get the correct result.

Step-6: Plot the middle line of the crop row

Center points between two edges were identified and plot the middle line. From top to bottom middle points of both edges were identified and drawn middle line of the crop row.

```
indices = np.where(edges != [0])
coordinates = zip(indices[0], indices[1])
start_point = (int((indices[1][0]+indices[1][1])/2),indices[0][0])
print(start_point)
end_point = (int((indices[1][len(indices[1])-1]+indices[1][len(indices[1])-2])/2),indices[0][len(indices[0])-1])
print(end_point)
color = (0, 255, 0)
thickness = 2
edgesMiddleLine = cv2.line(img, start_point, end_point, color, thickness)
```

Indices were stored on the edges point in white color and extracted all points using zip. The start coordinate point was stored in the start_point 2D array and the end coordinate point was stored in end_point 2D. The straight red line was plotted from the start to end-points of both edges line. Middle line points were stored in edgesMiddleLine array and plot green line on it. The middle line detection in the crop row is shown in Fig. 7.

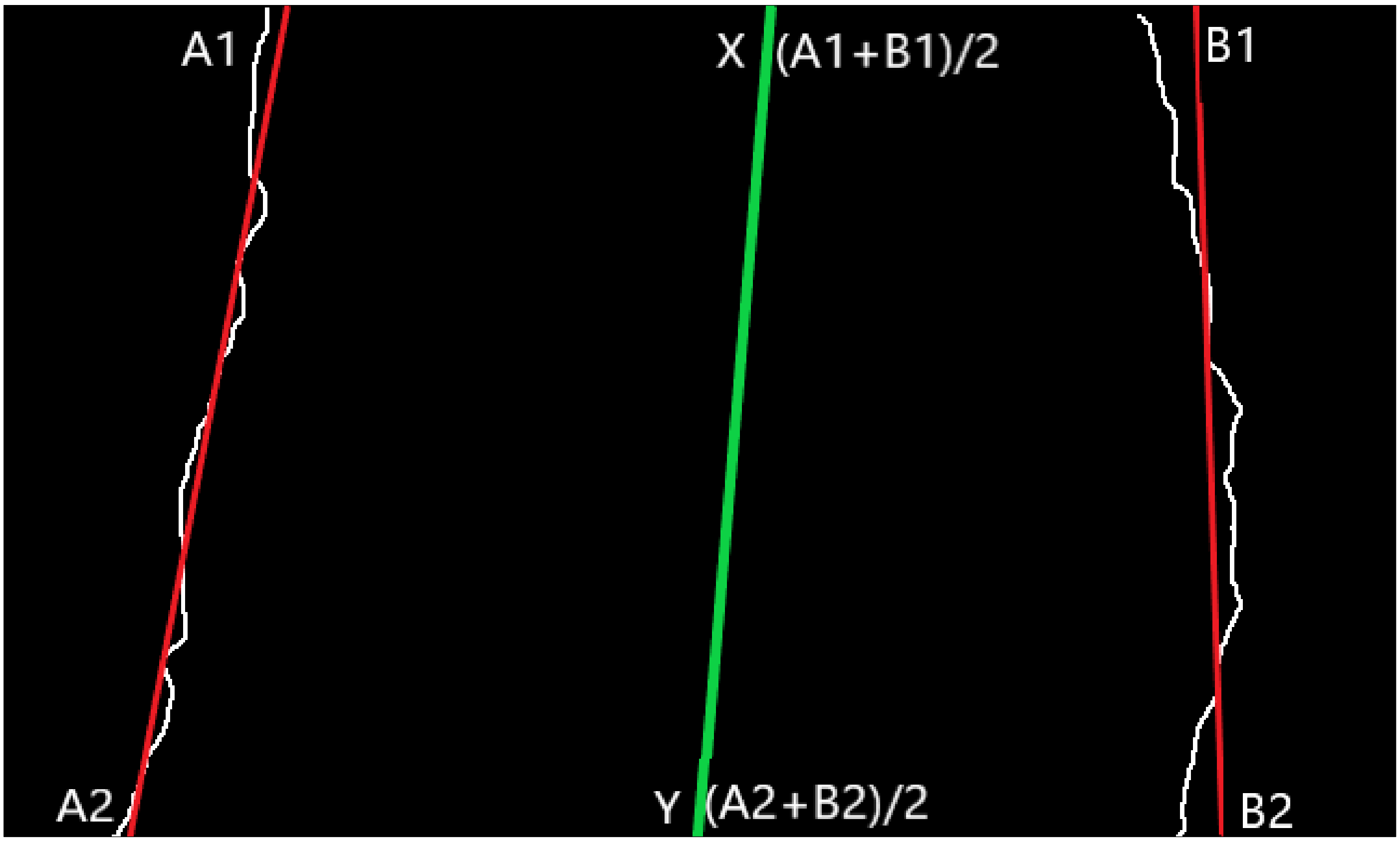


Fig 7: Crop row middle line plotting in image

In Fig. 7, the coordinate points of the top right and top left side edge detection were denoted as A1 and B1 and the coordinate points of bottom right and bottom left side edge detection were denoted as A2 and B2. The top and bottom center points of the middle line of the crop row were denoted as X and Y. Middle line of the crop row was drawn from extracting line between X and Y.

Step-7: Plot the centerline of the image

Machine deviation was observed when the machine deviates from the centerline of the crop row. The deviation was measured from the difference between the centerline of the crop row and the centerline of the image. The middle line of the crop row was identified in the previous step. While the centerline of the image was observed. The centerline of the image is shown in Fig. 8.

```
height, width, channels = img.shape
color = (0,0, 255)
centerUpper = (int(width/2),0)
centerLower = (int(width/2),int(height))
image = cv2.line(edgesMiddleLine, centerUpper, centerLower, color, thickness)
```

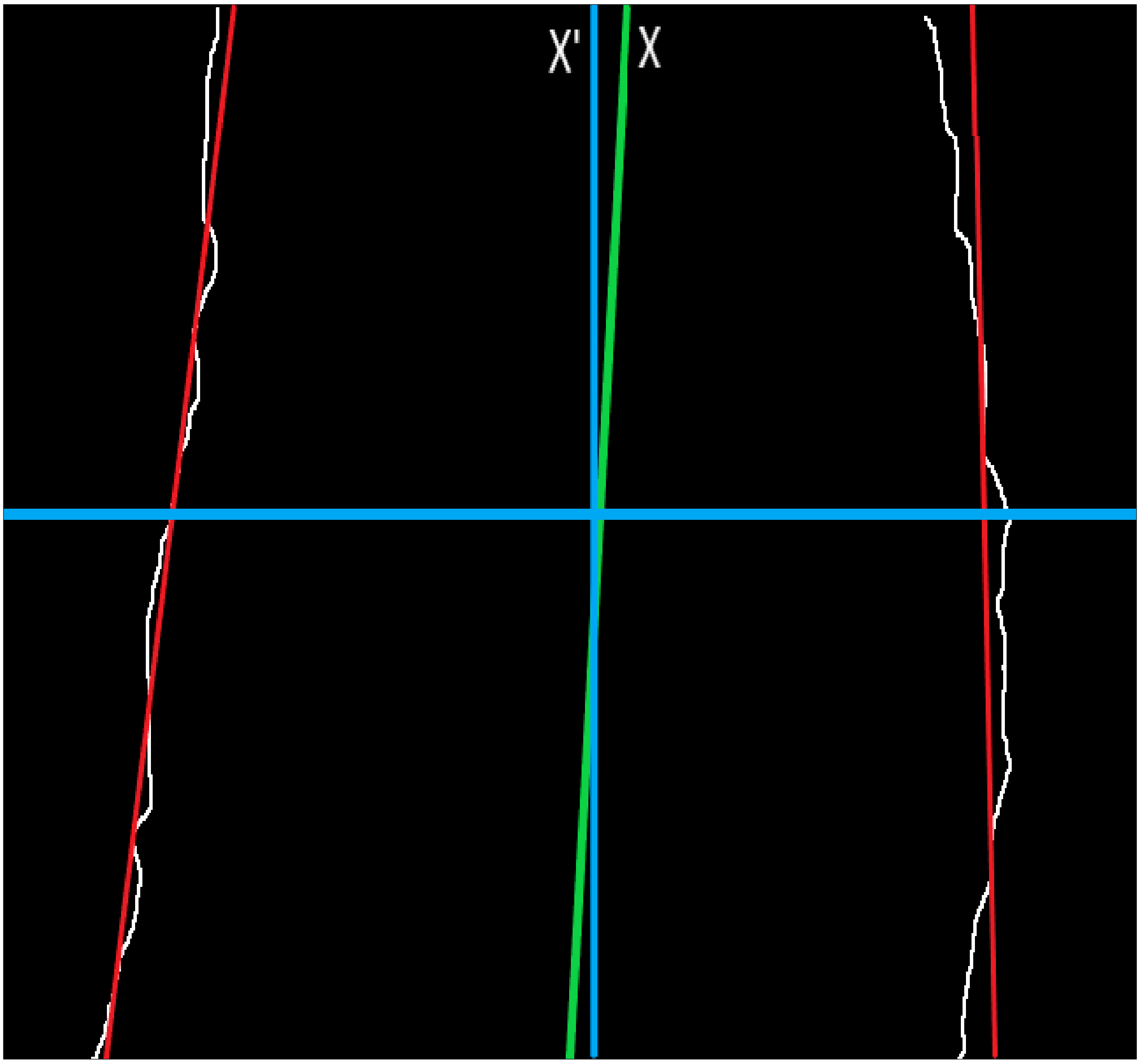


Fig 8: Image centerline and crop row middle line

In Fig. 8, X is a line that represents the middle line of two edge lines and X’ is a line that represents the centerline of the image in respective of height and width.

Step-8: Plot the angle of deviation

The angle between the crop row middle line and image centerline was observed. The deviation angle was measured using formula 3 and is shown in Fig. 9.

$$\tan\theta = XX' / (h/2) \qquad (3)$$

Where,
XX’ = Distance between two center points
h = height of the image

```
xPoint = start_point
xDPoint = centerUpper
midPoint = (centerLower - centerUpper) / 2
tanVal = (xPoint - xDPoint) / (xDPoint - midPoint)
(xPoint - xDPoint) = tanVal * (xDPoint - midPoint)
tanAng = math.tan(tanVal)
```

xPoint in program is a point X which is the crop row midpoint. xDPoint in program is a point X’ which is the midpoint of the centerline. O is midPoint which is the center point of the image frame. tanVal is tan formula value which is [Adjacent side/ Side opposite]. θ is tanAng which is angle deviation. XX’ is (xPoint - xDPoint) which is tracking lateral error.

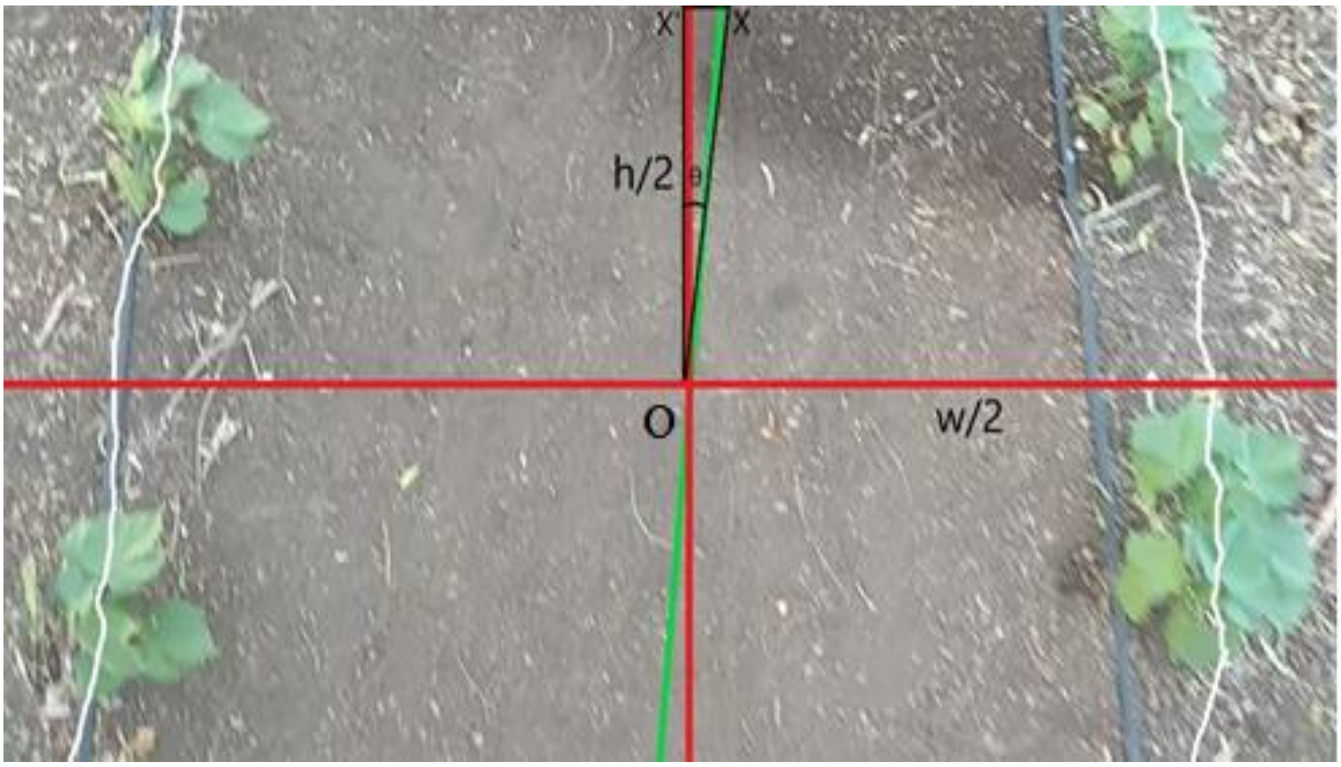


Fig 9: Angle of deviation from crop row middle line

Results and Discussion

The objective was fulfilled by developing an algorithm for the crop row identification and vehicle movements within the rows. That algorithm was developed in OpenCV software using Python programming language. In the image processing technique, different electronic components were used namely raspberry pi, webcams and relay module. The image processing technique was utilized for guiding the machine in between the rows of a crop. DC motors were mounted to the front wheels for easy steering. In the image processing technique, the cameras sensed the crop row and gave the command to the DC motors through raspberry pi and relay module. The deviation of the machine centerline from the crop centerline along with lateral error was calculated. The decision was taken by the microcontroller based on the predefined consideration points and lateral error. The machine was moved left or right to minimize the lateral error by changing the speeds of motors on the front wheels with the help of the relay module, which helped the machine to move straight in between the rows.

Conclusion

The minimum tracking lateral error, angle of deviation and reaction time were found to be 4.37 cm, 9.4° and 0.47 s for cotton crop and 3.35 cm, 7.01° and 0.38 s for okra crop respectively at camera height of 60 cm and camera angle of 45°. That was the best result in crop row detection techniques.

References

1. Doxygen. Canny edge detection; c2021. Retrieved from https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html
2. Doxygen. Depth map from stereo images; c2021. Retrieved from
3. https://docs.opencv.org/4.5.2/dd/d53/tutorial_py_depthmap.html
4. Fari MA. Study of image segmentation using thresholding technique on a noisy image. International Journal of Science and Research. 2013;2(1):49-51.