Using Computer Vision to Find the Equation of a Circle/Ellipse

Jake S. Walker
University of Hawai'i at Mānoa
jakesw@hawaii.edu

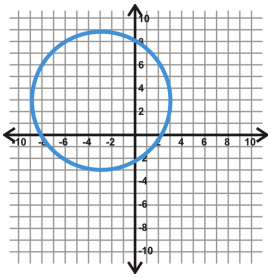
Abstract

This project shows the results of detecting a circle or ellipse in a grid graph and finding its equation using computer vision methods. The Hough transform is used as the baseline method for detecting the grid lines and axes in the graph. The same method and color detection are compared to determine the best approach for detecting the circle and ellipse. Finally, the project experiments with training a deep learning model to explore the possibility of expanding the project to detect quadratic curves beyond circles and ellipses, and find their equations.

1 Introduction

Educational technology has become an important part of students' lives. Digital learning tools such as video platforms, virtual classrooms, augmented reality, and robots have allowed for personalized learning and contributed to students' successful academic career [7]. One such tool is *GeoGebra*, a free and partially open-source 3D graphing calculator which has benefited me personally in learning multivariable calculus by providing a visual representations of quadratic surfaces.

Currently, there are many online graphing calculators that, given an equation, display the graph of the equation. They also have features that, given points and the type of quadratic curve, generate a best-fitting curve. However, these tools do not utilize computer vision methods to find the equation of a quadratic curve; in other words, the points and the type of quadratic curve must be identified manually, and entered into the tool to find the best-fitting curve. The goal of this project is to create a tool that takes an image of the graph of a quadratic curve as an input and outputs the equation of the curve (an example is shown in Figure 1). Further, by showing the steps to find the equation of a quadratic curve from a graph, students can gain a better understanding of how the equation is formulated from the graph.



$$(x+3)^2+(y-3)^2=6^2$$

Figure 1: An example of an input image (left) and its output solution (right)

Due to time constraints, the scope will be limited to finding equations from the graphs of circles and ellipses. However, a deep learning approach for finding the equations of quadratic curves beyond circles and ellipses will be explored and discussed later in this report. Lastly, the final working program for finding the equation of a circle or ellipse can be found in the following *GitHub* repository: https://github.com/jakeswalker/uhm-ics483-final-project.

2 Related Work

Although I was unable to find any related work on finding the equation of a quadratic curve using computer vision methods, there are many scientific paper that deal with identification of lines and geometric shapes such as circles and ellipses.

As described by Duda [1], the Hough transform is a computationally efficient procedure for detecting lines in an image. It achieves this by using a method of transforming the image space into a parameter space. Furthermore, as stated in Hassanein *et al.* [3], the Hough transform has evolved over the years to identify other arbitrary shapes, such as circles and ellipses. In this project, the Hough transform method for detecting grid lines and axes on a graph will be explored. This method, along with other approaches, will be compared to determine the best approach for detecting the circles and ellipses. Additionally, the technique for preprocessing described by Hassanein *et al.* will be used—specifically, the Canny edge detec-

tor.

Another computer vision method commonly used to detect circles, as described by Liu [4], is RANSAC. It is an algorithm that iteratively estimates the parameters of a mathematical model from a set of observed data, which may contain edges that deviate from the expected patterns. Since RANSAC is a non-deterministic algorithm, it produces a result with only a certain probability. While this approach is suitable for detecting imperfect circles, it is less effective for detecting curves such as circles and ellipses on a graph.

According to Nayyer *et al.* [5], color detection combined with machine learning techniques has gained popularity in recent years. They state that this is due to the object recognition algorithms that utilize colors are resistant to the influence of other properties like shape, surface, and settings. The color detection also makes it ideal for detecting curves on a graph as the curves are often plotted in color on a grayscale graph (as shown in Figure 1). For this reason, this project will also explore a straightforward yet powerful method of using pixel processing and color detection to detect circles and ellipses on a graph.

Utilizing deep learning methods for object detection is a key topic in computer vision, as many scientific papers have been written on this subject. One example is Ercan *et al.* [2], which discusses the limitations of the Hough transform for detecting circles, and explores various deep learning networks, including convolutional neural networks (CNNs), for efficient circle detection. This project will also employ a convolutional neural network (CNN) to predict the type of quadratic curve in a graph.

3 Data

3.1 Testing the Program

To stay within the scope of this project, the program that finds the equation of a circle or ellipse makes the following assumptions about the input image:

1. The graph is not handwritten
2. There is only one curve on the graph
3. The grid lines and axes are in grayscale, with the axes being darker than the grid lines
4. The curve is in color (non-grayscale)
5. The distance between each grid line is 1
6. The equation of the circle and ellipse is in the form $\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$, with $a, b, h, k \in \mathbb{Z}^+$
7. The graph does not include rotated ellipse

In total, 12 images of the graphs of circles and ellipses were created to test the program. The images include screenshots of graphs of circles and ellipses from the three most popular online graphing calculators, *Desmos*, *GeoGebra*, and *Mathway* (an example is shown in Figure 2), as well as some random images of graphs of circles found online. All 12 images used for testing the program can be found in the following directory: https://github.com/jakeswalker/uhmics483-computer-vision/tree/main/input_image

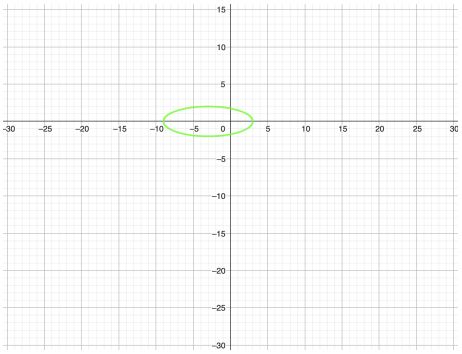


Figure 2: An example of an input image used for testing the program

3.2 Training the Model

To explore the possibility of expanding this project to detect quadratic curves beyond circles and ellipses, a deep learning model was trained to predict the type of quadratic curve in a graph. The training dataset includes 50 screenshots of graphs of circles from *Desmos*, and another 50 screenshots of graphs of parabolas from *Desmos*. Each curve was plotted using different coefficient values. An additional 5 screenshots of graphs were taken for both the circle and parabola classes for validating the model, and another 5 screenshots of graphs for both classes for testing the model. All images were resized to 224×224 in order to have a common input shape for the model (examples are shown in Figure 3). A total of 120 images used for training the model can be found in the following *Google Drive* folder: https://drive.google.com/drive/folders/1PpHGcztG55F8fpo5cnmbE1Zq9ZjDkeZN

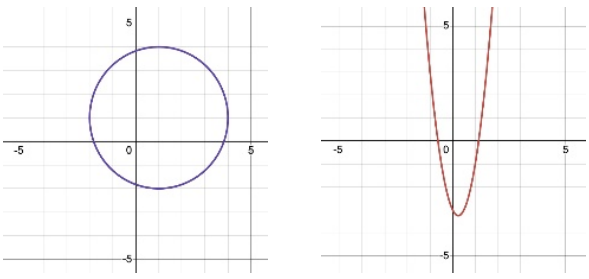


Figure 3: An example of a circle (left) and a parabola (right) used for training the model

4 Methods

4.1 Finding the Equation

In this section, the various steps taken in the program to detect the circle or ellipse in a graph using computer vision methods will be discussed. The program is written in Python and utilizes the OpenCV library; both of which are widely used tools in computer vision.

4.1.1 Detecting the Grid Lines

As stated in Section 3, the Canny edge detector is used to preprocess the input image. Prior to this step, the input image is converted to grayscale, which helps the Canny edge detector identify intensity gradients in the image. Once the image is preprocessed, the program applies the Hough transform to detect the lines in the input image, i.e., the grid lines.

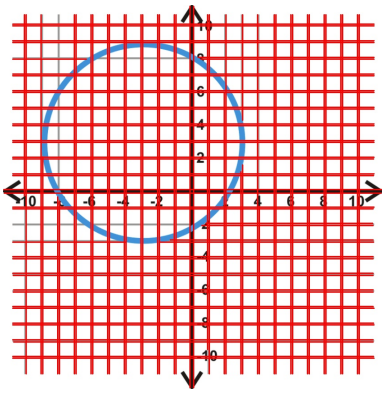


Figure 4: The red lines shows the detected lines in the input image after applying the Hough transform

As shown in Figure 4, the Hough transform detects multiple lines for each grid line. For example, two lines are detected for each axis. These redundant lines are removed in the next step by calculating the distance between each grid line, call it *unit_distance*, and starting from the leftmost vertical line, iteratively removing the adjacent lines that are within the *unit_distance* from each vertical grid line. The same procedure is repeated for the horizontal lines. Figure 5 shows the results of the image after removing the redundant lines.

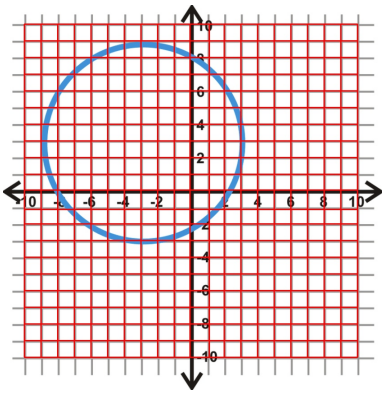


Figure 5: The redundant lines are removed and new lines are drawn for each grid lines

4.1.2 Finding the Vertices

Note that to find the pixel locations of the vertices of a graph, it suffices to know the x value of each vertical line, and the y value of each horizontal line. Once these values are obtained, the program finds each vertex in the graph through an iterative process, as shown in Figure 6.

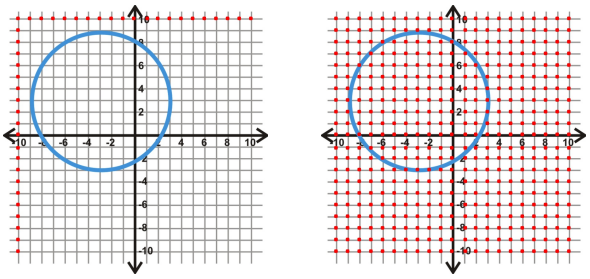


Figure 6: The x and y values of the lines (left) are used to find the vertices of the graph (right)

4.1.3 Finding the Origin

To find the origin of the graph, the program uses the RGB values of the pixels to identify the darkest grid lines, i.e., the axes of the graph. This requires the assumption that the axes are darker than the other grid lines, as discussed in Section 3.1. The left image in Figure 7 shows the region, highlighted in light blue, that the program scans to detect the axes. Once the axes are identified, the location of the origin is found using the locations of the axes. The origin of the graph is shown as a blue dot in the right image of Figure 7.

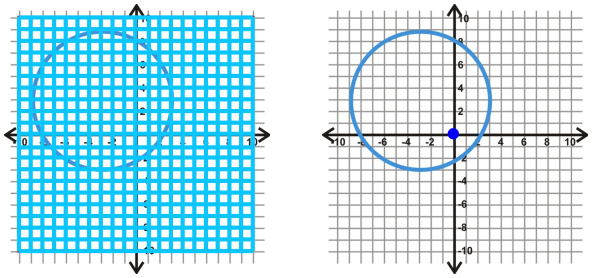


Figure 7: The program scans the light blue region (left) to find the origin of the graph (right)

4.1.4 Mapping the Graph

At this stage of the program, the locations of the vertices are stored as pixel locations (e.g., the pixel location of the origin of the graph shown in Figure 7 is (415, 410)). Thus, to find the equation of the circle or ellipse, the pixel locations need to be mapped to the coordinates of a unit-distance graph, where distance between each grid line is 1. This is done by, first, storing the locations of vertices in a one-dimensional list, call it *vertices_list*, in column-major order. The index of the origin is then extracted from the *vertices_list*, and stored as *origin_index*. Then for each vertex in *vertices_list*, the x coordinate of the vertex is calculated by

$$x = floor(i/h) - floor(origin\_index/h),$$

where i is the index of the vertex in *vertices_list*, and h is the number of horizontal grid lines. Similarly, the y coordinate of the vertex is calculated by

$$y = (origin\_index \ \% \ h) - (i \ \% \ h).$$

Figure 8 shows the result of mapping the graph to the unit-distance graph, with each vertex displaying its x coordinate.

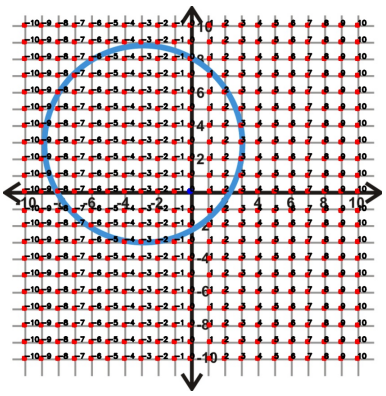


Figure 8: The pixel locations of the vertices are mapped to the coordinates of the unit-distance graph

4.1.5 Detecting the Curve

As discussed in Section 2, the Hough transform and color detection are compared to identify the best approach for detecting a curve on a graph, as outlined in Section 5. Since the Hough transform can easily be implemented using OpenCV's *HoughCircles* function, this section will discuss the implementation of the pixel processing and color detection methods for detecting the curve.

The color detection method utilizes the RGB values of the unprocessed input image. This requires the assumption that the circle or ellipse is in color and that the grid lines are in grayscale, as discussed in Section 3.1. Detecting the circle and ellipse is done in the program by evaluating the differences in the values of each RGB component. For example, the blue curve shown in the graph in Figure 1 has an RGB value of (85, 145, 210), the grid lines have RGB values of (149, 150, 147), and the axes have RGB values of (26, 24, 25). Then taking the sum of absolute difference of each RGB component, we have

$$difference\_sum = |R - G| + |R - B| + |G - B|.$$

We see that the RGB of the pixels of the curve have the largest *difference_sum*. Using this value, the program iterates through each pixel of the image and finds the pixels where the sum of absolute difference of each RGB component is close to the largest *difference_sum*, which are the pixels of the curve. Figure 9 shows the result of the described color detection.

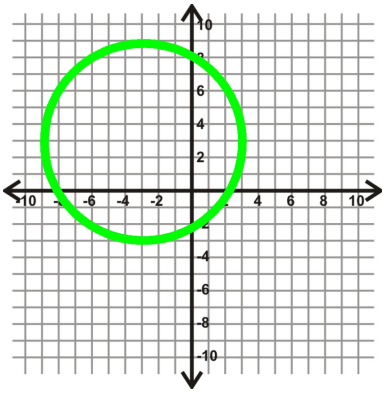


Figure 9: The detected curve is shown in green

4.1.6 Mapping the Curve

Once the pixel locations of the curve are obtained, the program finds the locations of the topmost and bottommost pixel of the circle or ellipse using the y values of the pixels. Similarly, the leftmost and rightmost pixels of the circle or ellipse are found using the x values of the pixels. The program then scans a region around the pixels to find the corresponding point in the unit-distance graph, as shown as the light blue region in Figure 10. The center of the circle or ellipse is then easily found using the mapped points.

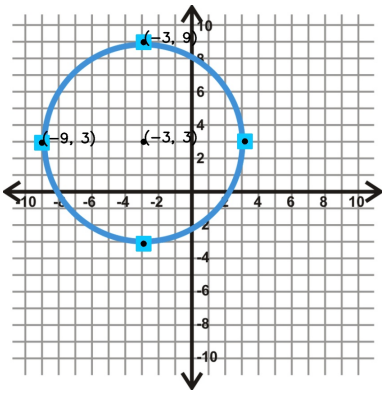


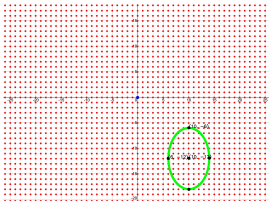
Figure 10: The pixel location on the circle are mapped to the unit-distance graph

4.1.7 Formulate the Equation

As seen in Figure 10, we now have all the necessary values to formulate the equation of the circle or ellipse in the form

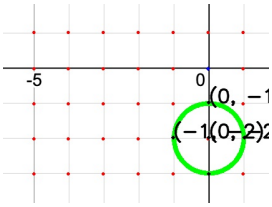
$$\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1.$$

Figures 11, 12, and 13 show the outputs of the program from three different inputs. All outputs from the 12 test images can be found in the following directory: https://github.com/jakeswalker/uhmics483-final-project/tree/main/output



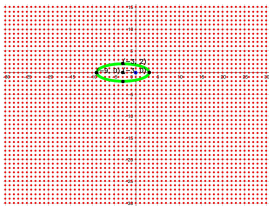
$$\frac{(x-10)^2}{4^2} + \frac{(y+12)^2}{6^2} = 1^2$$

Figure 11: The output (left) and the solution (right) of circle_08.jpg



$$x^2 + (y+2)^2 = 1^2$$

Figure 12: The output (left) and the solution (right) of circle_09.jpg



$$\frac{(x+3)^2}{6^2} + \frac{y^2}{2^2} = 1^2$$

Figure 13: The output (left) and the solution (right) of circle_03.jpg

4.2 Identifying the Quadratic Curve

As seen in Section 4.1, the program's algorithm for finding the equation of a circle or ellipse only works for these two shapes. Furthermore, using pixel processing to find the type of quadratic curve is inefficient and time-consuming. As Ercan *et al.* suggest, deep learning has recently dominated the computer vision field and its performance has been shown to be superior to classical computer vision methods. This section will explore the deep learning method for identifying the type of quadratic curve—specifically, identifying circles and parabolas. The implementation of the model training is found in the following *Google Collab Notebook*: https://colab.research.google.com/drive/126KB65kBewd0Sv8WFmIUez6GPzhh7VS9?usp=sharing

As discussed in Section 2, a CNN is employed to predict the type of quadratic curve in a graph—specifically, PyTorch's pre-trained model, EfficientNet, which is designed to achieve image classification tasks with fewer computational resources compared to other deep learning models (Tan and Le[6]).

After training the model with 50 images for each class (i.e., circle and parabola), the results showed that the training loss decreased with each epoch, suggesting that the prediction error decreased for each complete iteration over the training dataset. Similarly, validation loss decreased with each epoch, suggesting that the difference between the predicted values and the ground truth decreased for each complete iteration over the training dataset. Figure 14 shows the line chart of the loss over epochs

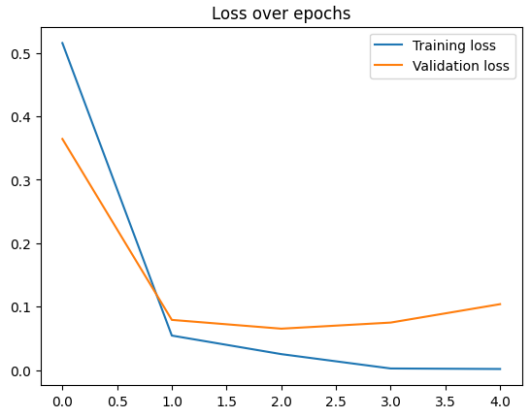


Figure 14: The training and validation loss over epochs

The prediction results showed that the model accurately predicted the test images, with circles having a probability of 1, and the parabolas having a slightly lower probability, as shown in Figure 15. These results show that the program can efficiently identify the type of quadratic curve using deep learning methods, and apply an algorithm specific to each quadratic curve to

find the equations.

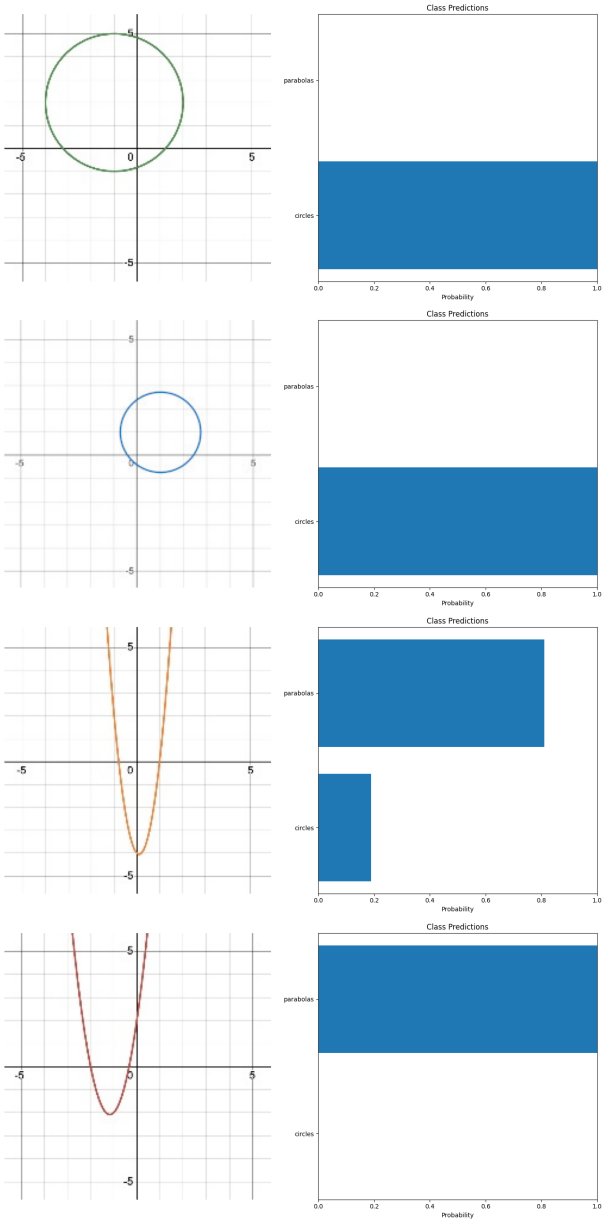


Figure 15: Class predictions of circles and ellipses

5 Experiments

As stated in Section 4.1.5, this section compares two methods for detecting a circle or ellipse in a graph: the Hough transform and color detection.

5.1 Color Detection Method

5.1.1 Limitations

As shown in Section 4.1.5, the pixel processing and color detection methods successfully detect circles and ellipses from all 12 input images. However, aside from the fact that this method does not work on grayscale curves, there are examples where it fails to detect a curve properly in an image, even when the image adheres to the assumptions made in Section 3.1. Figure 16 shows a screenshot of a curve from *GeoGebra* where the circle is labeled as *f*. The letter have different RGB values than the curve, and have the largest sum of absolute difference of each RGB component, thus making the algorithm to mistake it for the curve.

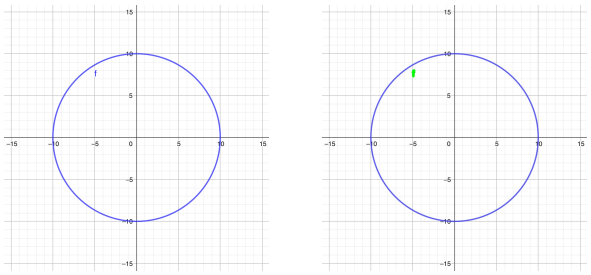


Figure 16: The color detection method mistakes the letter f as the curve

In addition, the pixel processing and color detection methods are highly inefficient due to their reliance on multiple iterative procedures.

5.1.2 Benefits

As mentioned by Nayyer *et al.*, one of the benefits of using the color detection method is its resistance to the influence of other properties, such as shape. Figure 17 shows the color detection method detecting the hyperbola, using the same algorithm that was applied to detect a circle.

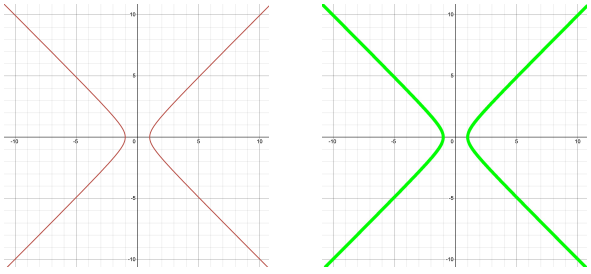


Figure 17: The color detection method can detect the hyperbola

5.2 The Hough Transform

5.2.1 Limitations

As stated in Ercan *et al.*, the Hough transform for detecting circles requires three parameters (x, y, r) instead of two (r, θ) for detecting lines. This results in the algorithm requiring more memory space

and higher computational complexity, making the processing speed slow.

As for detecting ellipse, the Hough transform requires a different algorithm from detecting circles. According to Hassanein *et al.*, the Hough transform requires five parameter for detecting ellipse, making the computational complexity even higher. Furthermore, the OpenCV library does not have a built-in functions like *HoughCircles* for detecting ellipse, making it more challenging for programmers to implement ellipse detection. Figure 18 shows OpenCV's *HoughCircles* function applied on an ellipse.

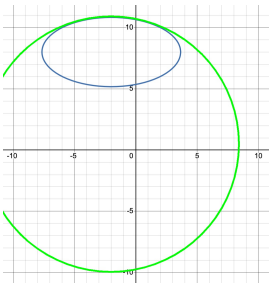


Figure 18: OpenCV does not have built-in support for ellipse detection using the Hough transform

5.2.2 Benefits

The main benefit of using the Hough transform over color detection is that it does not rely on the assumption that the circle or ellipse is in color. Furthermore, the Hough transform is more robust to noise as it can detect the circle that the color detection method failed to detect. The left image in Figure 19 shows a grayscale circle, and the right image shows the circle with the letter f next to it.

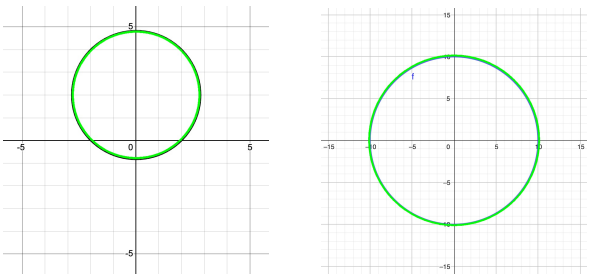


Figure 19: The Hough transform can detect the circles that the color detection method failed to detect

6 Conclusion

This project demonstrated that the equation of a circle or ellipse can be found using computer vision methods. Furthermore, it showed that the type of quadratic curve can be efficiently identified using deep learning methods, suggesting that the equations of other quadratic curves, beyond circles and ellipses, could be found with further implementation. It also showed that both the Hough transform and color detection have their respective limitations and benefits in detecting circles or ellipses on a graph.

For future improvements, some of the assumptions about the input image could be eliminated, such as the requirement that the input image not be a handwritten graph, or that the distance between each grid line be 1. The algorithm for color detection could be improved, as the current implementation requires a lot of computational resources. Furthermore, collecting more data could improve the accuracy of predictions for the type of quadratic curves, and would allow for identifying different types of quadratic curves.

The main takeaway from this project was that there are limitations to classical computer vision methods for detecting shapes and objects, and that deep learning methods dominate the field and are generally superior to traditional methods.

References

[1] Richard O. Duda and Peter E. Hart. Use of the hough transform to detect lines and curves in pictures. *Graphics and Image Processing*, 15(1), 1972.

[2] M. Fikret Ercan, Allen Liu Qiankun, Simon Seiya Sakai, and Takashi Miyazaki. Circle detection in images: A deep learning approach. *IEEE OCEANS*, 2020.

[3] Allam Shehata Hassanein, Sherien Mohammad, Mohamed Sameer, and Mohammad Ehab Ragab. A survey on hough transform, theory, techniques and applications. *IJCSI International Journal of Computer Science*, 12, 2015.

[4] Song Liu. Fast and accurate circle detection algorithm for porous components. *Journal of Electrical Engineering Electronic Technology*, 3(1), 2014.

[5] Aadarsh Nayyer, Abhinav Kumar, Aayush Rajput, Shruti Patil, Pooja Kamat, Shivali Wagle, and Tanupriya Choudhury. Color-driven object recognition: A novel approach combining color detection and machine learning techniques. *EAI Endorsed Transactions on Internet of Things*, 10, 2024.

[6] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *International Conference on Machine Learning, 2019*, 2020.

[7] American University. How important is technology in education? *School of Education Online*, 2024.