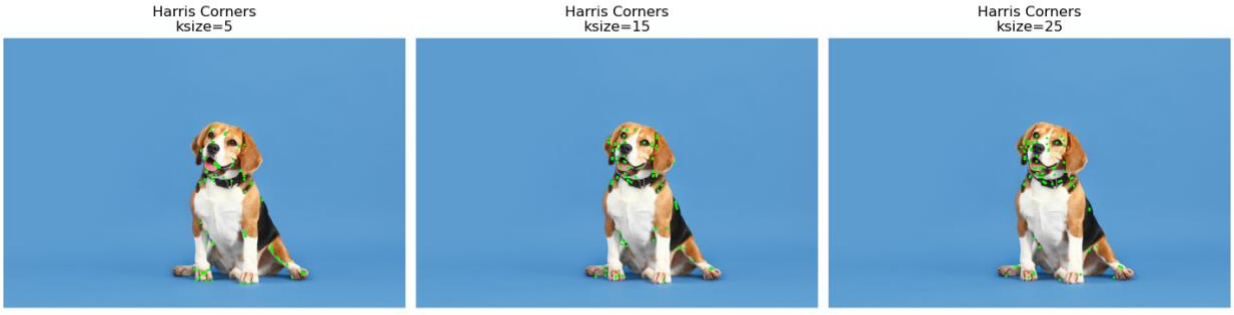
Edge Detection

The initial stage of my image analysis project examined the effectiveness of edge detection approaches among Sobel, Laplacian and Canny when analyzing an apple photograph. The program started with necessary processing procedures that converted images to grayscale while using Gaussian blur for noise reduction to establish a stable platform before edge detection. In the context of this specific apple image, the Sobel edge detection method provides a valuable balance between shape delineation and detail preservation. Its ability to capture subtle surface textures, coupled with its production of continuous edges, makes it a highly effective choice. While Canny offers cleaner, thinner edges, Sobel's output is more informative for applications that require a comprehensive representation of the object's features. The Laplacian detection method found rapid changes in intensity yet generated numerous irrelevant signals in addition to the detected edges.

Corner detection



The analysis of the Harris corner detection algorithm through systematic testing examined the effects that KERNEL_SIZE has on its performance. I processed a dog image while implementing Harris corner detection using different KERNEL_SIZE values from 5 to 15 to 25 to visually assess

the generated results. Through this methodology I monitored how KERNEL_SIZE affected the Harris corner detection process and its precision when identifying corners.

The experimental data shows that lowering the KERNEL_SIZE value enables the detection of greater corner numbers to identify finer details within the image. Bigger KERNEL_SIZE values create a system that finds fewer but more dominant corners which pick up strong intersection points in images. The controlled changes to KERNEL_SIZE provided effective insights about the relationship between corner robustness and detail preservation.

Line and Circle detection

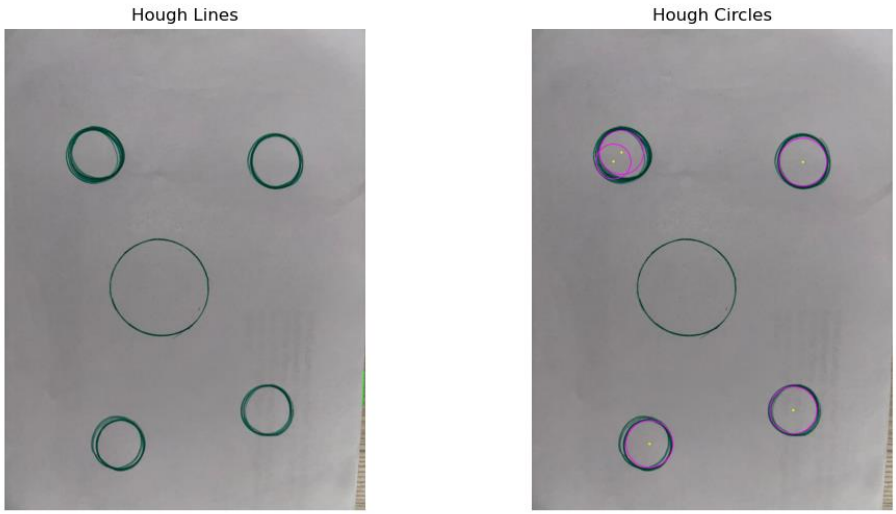
The Hough transform served as the basis for my implementation to detect lines and circles in bicycle images during this phase of project development. The essential function of parameter optimization for Hough transform demanded a step-by-step threshold adjustment to achieve peak detection precision.

I used probabilistic Hough transform (cv2.HoughLinesP) to detect lines because it yielded high precision in line detection of bicycle frame elements. The detection performance was optimized through precise configuration of threshold and minLineLength and maxLineGap parameters.

With the Hough gradient method (cv2.HoughCircles) I detected circles by adjusting dp, minDist, param1, param2, minRadius and maxRadius parameters to match the bicycle wheel dimensions with other circular components. The successful precision of circle detection depended on this process of performing repeated parameter modifications.

This image in the first example was taken from the internet. I implicitly took this photo because it was the exact match for the assignment requirement. In addition I wanted to experiment

with the photo taken from my phone but none of them performed well.
For the proof I want to illustrate one example:



I drew some circles with the pen on the paper. The bold circles detected after threshold configurations, but the slimmer lines could not be detected.

The lightening and perspective affect the results the most. That's why most of my result analysis are focused on professional images. You can also find the smarties.png file that is perfectly offered for this task by OpenCV itself.

Interest Point detection (ORB)

For this part of my image analysis, I wanted to see how well we could match features between an image and a version of it that had its perspective changed. Basically, I created a 'warped' version of the image using `cv2.getPerspectiveTransform` and `cv2.warpPerspective`. To do this, I carefully picked points in the original image and decided where I wanted them to move in the warped one. This let me control exactly how the image's viewpoint changed, so I could test how well our matching would work under these conditions.

Then, I used the ORB algorithm – it's pretty fast and good at finding features even when things change a bit in the image. ORB gave me these 'key points' and 'descriptors' that help identify unique parts of both the original and warped images. I limited it to 1000 features, just to keep things efficient.

Next up, I matched these features using a 'Brute-Force' method with something called 'Hamming distance'. I also used a 'cross-check' to make sure the matches were really solid. Then, I sorted the matches by how similar they were, with the best matches coming first.

To see how well it worked, I drew lines connecting the top 50 matches (this also can be adjusted) on a combined image. This gave me a visual idea of how accurate the matching was, even with the perspective change.

I also calculated the average 'distance' between these matches for quantitative analysis. A lower distance means the matches are better. In this case, it was 16.18, which is pretty good! I saved this number to a text file for later.