WIC Self Learning

Oregon State University

ENGR 416 D-601

Autonomous Low-Speed EV Platform Development

Camden Galen

April 19, 2026

1. Introduction

For this self-learning assignment, I chose to learn the basics of computer vision for camera-based line detection using Python and OpenCV. I selected this skill because it directly relates to one of the core functions of my autonomous low-speed EV platform development capstone project. As described in the design proposal, the low-speed EV (LSEV) must be able to reliably travel between Oregon State Universities machine shops for material transport, and the proposed solution is a line-following and collision-detection system. To address the line-following portion of the project, the open-source Python library OpenCV can be used to identify the path the EV should follow [1]. This skill develops design attributes related to sensing, software integration, prototyping, and control-system preparation.

To start my learning, I worked through a YouTube tutorial series focused on basics of OpenCV and line-detection concepts [2]. After that, I built a Python program that can detect a line when given a single image. This was completed with help from Codex, which assisted with scaffolding the initial Python and OpenCV code, improving code comments, troubleshooting detection behavior, and organizing the finished work in a GitHub repository in a timely manner [3, 4]. Even with that support, I still had to choose the approach, run the code locally, test different images, and evaluate whether the results were useful for my project.

2. Skill Application

The new skill I practiced was implementing a basic computer-vision pipeline for line detection. After working through the tutorial material, I wrote a Python script that loads an image, converts it to grayscale, applies Gaussian blur, performs Canny edge detection, and then uses the Hough transform to identify line segments. The final output overlays the detected line segment on the original image, which makes the result easy to visualize and evaluate.

I tested the script on multiple images that I took myself. My first test case used a photo of a white phone charger placed on a dark desk. This image acted as a more realistic and slightly more challenging test because the scene included background texture, lighting variation, and other objects. Even with that added complexity, the script was still able to detect a usable segment of the cable. While the result was not as clean as a controlled test image, it still showed that the method can work in a less ideal environment.

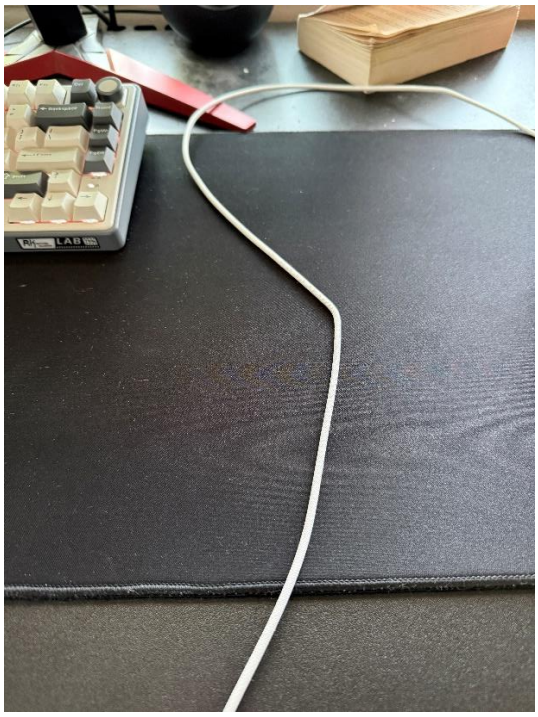

Figure 1: White phone charger with black desk background test image.

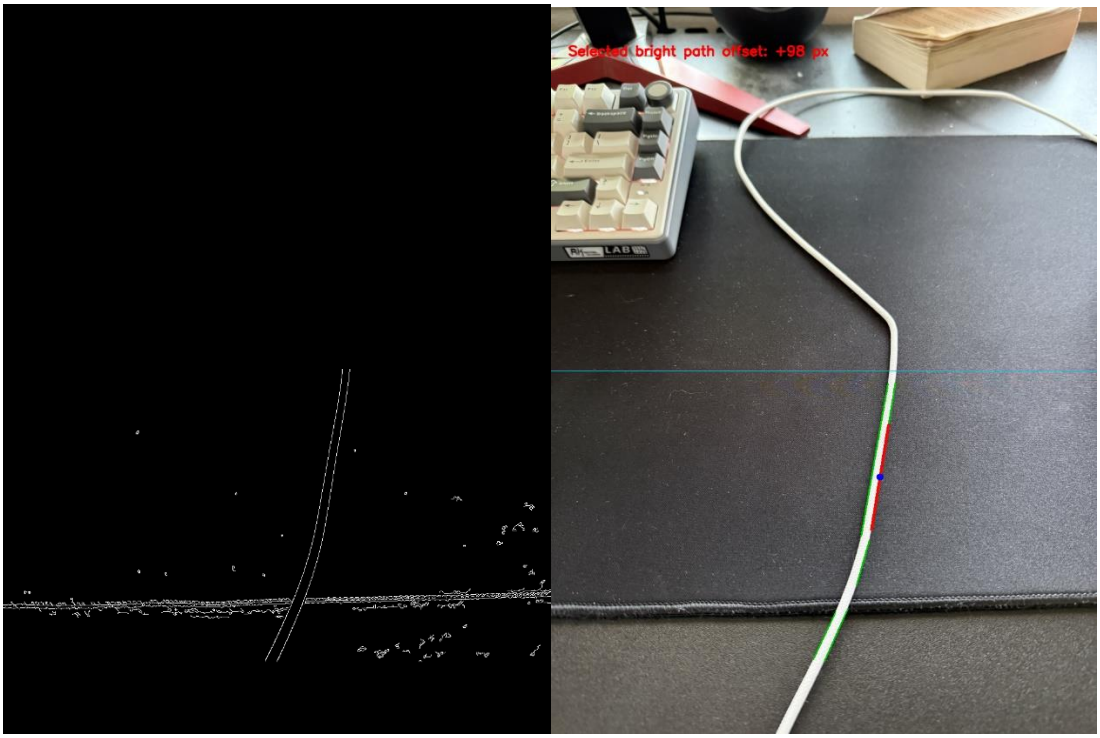

Figure 2 and 3: Phone charger line output

My second test case used a black Sharpie line drawn on white paper. This created a more controlled image with stronger contrasts and very little background noise. As expected, this test produced a cleaner result and made it easier to see how each step of the vision pipeline contributed to the final detection. The detected line segment aligned more clearly with the visible path, which made this example useful for understanding the strengths of the approach.

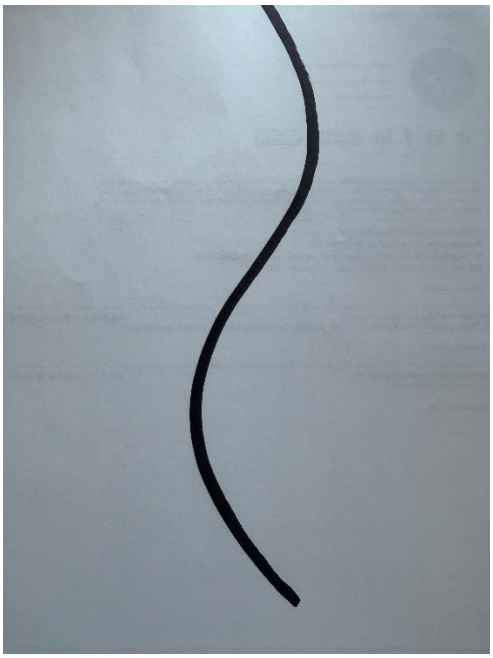

Figure 4: Sharpie line on paper.

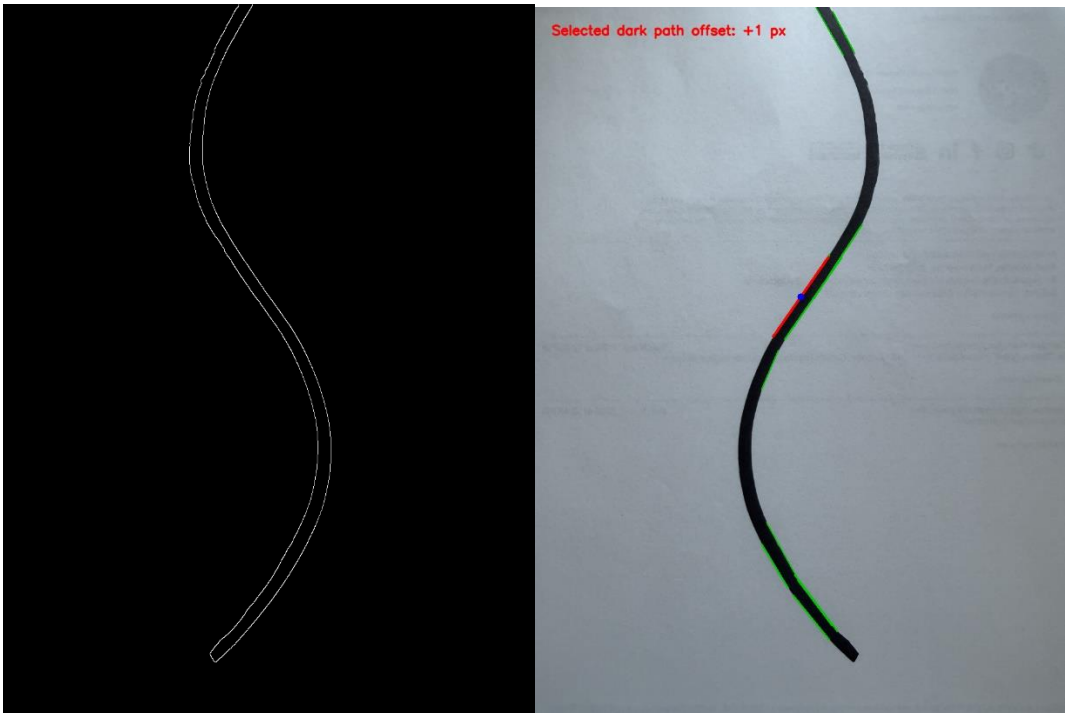

Figure 5 and 6: Sharpie line output.

Comparing the two test cases helped me better understand how image quality affects computer-vision performance. The charger image showed that the algorithm can still produce a useful output in a more realistic scene, while the Sharpie image showed that a high-contrast and low-noise environment leads to a cleaner and easier-to-interpret detection result.

3. Results

The final prototype successfully demonstrated the basic concept of camera-based line detection using only my computer. The script was able to detect a visible path from static images and highlight a selected line segment on the output image. In the charger test, the algorithm detected a useful segment of the cable despite the added clutter in the scene. In the Sharpie test, the result was cleaner and more stable because the image had much stronger contrast. Together, these examples showed that the approach can work in both controlled and more realistic conditions, although image quality has a clear impact on performance.

The script also calculated a pixel offset between the detected line segment and the center of the image. That offset is important because it could eventually be used as an input for steering correction in the LSEV platform. One of the more important lessons from this exercise was that the Hough transform detects straight line segments rather than full curved paths. In this case, that was acceptable because the goal was to learn the underlying process and apply it to my project, not to create a fully optimized production system.

This work gives me a useful starting point for future capstone development. The next step would be integrating the code into a live webcam feed so the system can process frames in real time. From there, the line-detection output could be combined with obstacle-detection logic as part of a larger autonomous navigation system.

4. Conclusion

This self-learning activity helped me independently develop a new technical skill that is directly applicable to my capstone project. I learned the fundamentals of OpenCV-based line detection and applied them by building a working prototype, testing it on multiple images, and evaluating the strengths and limitations of the results. The experience improved my understanding of how a camera-based line-following system could work on an autonomous low-speed electric vehicle and gave me a realistic foundation for future software integration.

Resources

[1] opencv-python, Python Package Index. Available: https://pypi.org/project/opencv-python/

[2] YouTube, “OpenCV tutorial series,” video playlist.
Available: https://www.youtube.com/watch?v=zSa-fOGh8es&list=PLlD0XVjVhLaLVZWgJuOBrv4JBsWK99DGV&index=1

[3] OpenAI Codex, AI coding assistant, used to assist with Python code scaffolding, debugging, and repository organization, April 2026.

[4] C. Galen, “line-detection-lsev,” GitHub repository. Available: https://github.com/camdengalen/line-detection-lsev