Homographies for Computer Vision

VCOM - Project 1

Carlos Gomes, FEUP - up201906622@edu.fe.up.pt
Filipe Pinto, FEUP - up201907747@edu.fe.up.pt
Pedro Nunes, FEUP - up201905396@edu.fe.up.pt

April 2023

1 Introduction

Homographies are a fundamental concept in the field of computer vision, establishing a connection between points on a planar surface and their corresponding counterparts in an obtained image. Numerous real-world uses for this relationship may be found, and that's exactly what we are going to investigate in this project. We will take a look at the effectiveness of homographies in some tasks, including localizing known objects, calculating plane distances, identifying objects using template matching, and creating a bird's-eye perspective of a lane. Through these tasks, we will learn more about homographies and their crucial function in computer vision.

2 Localize Objects

2.1 Selecting Images

To implement this application four pairs of images were used: "Where's Wally?" example, a cookie box, an elephant statue, and a staples remover box and their respective scenes. Different objects with different shapes, sizes, textures, and testing planar and non-planar objects and scenes with different backgrounds and lighting conditions will help us testing the algorithm with all these distinct conditions. To explain the process, we will use the elephant example to show the results.

2.2 Apply Grayscale

The first step was to convert the images to grayscale. It is a usual process in feature detection to reduce the color information to just shades of gray, which makes it easier to detect edges and other relevant features since they only have one channel of information, as opposed to color images, which means that they require less computational resources to process.

2.3 KeyPoints and Descriptors Detection

The next step, and one of the most important for this application, is keypoint detection. To do this, 4 algorithms were tested:

- **SIFT**[8]: The Scale-Invariant Feature Transform detects, describes, and matches image features despite variations in size, orientation, and perspective. It is robust to scaling, rotation, and affine distortion. Results for this algorithm can be found in Figure 1.
- **ORB**[7]: Oriented FAST and Rotated BRIEF combines FAST keypoint detector and BRIEF descriptor by utilizing keypoint orientation to construct a rotation matrix for the descriptor. Quick speed, high performance, feature extraction ability, and numerous patents are some characteristics of it. Results for this algorithm can be found in Figure 2.
- **BRISK**[2]: algorithm with scale invariance and rotation invariance. It has high-quality performance as in state-of-the-art algorithms and creates feature descriptors by utilizing grayscale relationships between random

point pairs in the local image neighborhood to construct local feature descriptors. Results for this algorithm can be found in Figure 3.

- **KAZE**[5]: works by detecting keypoints using a non-linear scale space and then describing these keypoints using a set of descriptors that are similar to those used in SIFT. Results for this algorithm can be found in Figure 4.

2.4 Matching

After finding the feature points it was necessary to match them, so **knnMatch** was used [6]. Then to remove some of the outliers existing in the found matches we applied the **Lowe Ratio Test**. Some of the results can be found in Figure 5, Figure 6, Figure 7 and Figure 8. One interesting finding for this specific image is that a ratio somewhere between 0.37 and 0.38 is the limit, which means that any ratio below that value will not provide any matches between both images. Then the **RANSAC** method was applied. Due to noise or outliers in the input data, the matching algorithm without the RANSAC (Figure 9)approach may provide inaccurate correspondences. For this specific case, the RANSAC is used to calculate the homography transformation, which is then utilized to determine the position and orientation of the object in the scene image. So RANSAC can significantly reduce the number of false positives in the object detection results, which is crucial for many applications such as object localization, which matches this application.

2.5 Conclusions

To sum up, feature detection has been successfully applied to the task of localizing objects across various images. The Lowe Ratio test, RANSAC, and experimenting with various feature detection algorithms were all combined to successfully complete the application.

3 Measure positions and distances on a plane with a single camera

3.1 Select Images

This application was developed using two images from different perspectives. Each picture depicts a plane with a 150-millimeter-side square. To use these images correctly, we had to converter from BGR (default format used by opencv when reading an image) to RGB. These images will help us to test the program developed.

3.2 Homography calculation

Homography is a transformation that maps points in one image to corresponding points in another image or plane. In order to calculate the Homography that relates the points on the plane and on the perspective image we need to select the interest points in the image. For that, we must select six points, manually, in the image. The points must be chosen clockwise, first the four vertices of the square, starting from the top left corner, and then the two points marked inside the square whose coordinates are well known. Then, we need the real word coordinates of the selected points in order to compute the homography matrix that relates the points on the plane and on the image. In general, at least four corresponding points are required to compute the homography matrix, however in some situations, four points might not be enough to accurately describe the transformation. In order to provide a more precise and reliable homography estimation, six or more comparable points are frequently employed. By selecting the six points, the extra points provide more restrictions, which increases the accuracy of the solution. In order to deal with outliers and further increase the accuracy of the homography matrix, we also apply a robust estimator **RANSAC** (Random Sample Consensus). To analyze the effect of using the RANSAC method or not, we can compare the accuracy of the distance measurement with and without RANSAC for the same image. Without RANSAC, the homography matrix can be estimated using all the correspondences, including the wrong ones. This can lead to inaccurate distance measurement, especially when the number of wrong correspondences is high (Figure 12, 13 and 14). On the other hand, using RANSAC can eliminate the effect of wrong correspondences and lead to a more accurate distance measurement (Figure 10 and 11).

3.3 Calculate the position on the plane

In order to calculate the real coordinates, we define a function which takes a point defined by its pixel coordinate and a homography matrix as inputs and returns the real-world coordinates of the corresponding point in the transformed image. The function uses OpenCV's *perspectiveTransform* function to transform the pixel coordinates of the point to the real-world coordinates using the homography matrix. We round the numbers to one decimal places in order to simplify their representation and make them more readable in the perspective image.

3.4 Calculate the real distance (in mm) between points on the plane

Overall, allows the user to interactively select points in an image and visualize the distance between the last two selected points in real-time. In order to calculate the real distance between points, we use the **Euclidean** distance formula between the real-word coordinates of the points. We could use other distance metrics like **Manhattan** or **Chebyshev** but **Euclidean** is preferred because it is more appropriate for measuring distances between points in a continuous space which is the case.

3.5 Conclusions

In summary, with the use of basic methods like point selection, coordinate transformation, and homography computation, we were able to develop an image processing application that lets users calculate distances in an image. The project highlights the importance of understanding basic image processing concepts like **homography** and **RANSAC**, and demonstrates how these techniques can be used to create useful applications. Overall, the project successfully achieved its objective of creating an application that can measure real positions and real distances in an image, and serves as a good example of the power of image processing and computer vision techniques in solving real-world problems.

4 Recognize planar markers using template matching

4.1 Selecting Images

For this application testing, two images were used. They consisted in a busy environment, with one image having a frontal view of the markers and other with a perspective view. For template matching, two different markers were used (Figure 15).

4.2 Corner Detection

Both a manual and non-manual corner detection were approached. For the manual approach we're required to select the corners of both markers. It yields good results, but it requires user interaction and it really depends if the corners were properly chosen. For the non-manual corner detection, we first tried with harris corner detector. However it was yielding poor results, since there were a lot of corners and figures in the image. Thus, we decided to use SURF for key points detection.

4.3 Homography Calculation and Frontal View Transformation

After we have the corners of both squares, we calculate the homography and respective frontal view transformation for both of them. This makes our application robust to perspective distortions. Following the homography, we then apply a warp perspective, so to get a frontal view of each square. After the warp perspective we apply template matching between the frontal view of the markers and the template images. For template matching we use a correlation coefficient. There is a different threshold between manual and non-manual corner selection, to account for possible human errors. If the frontal view of the markers matches with the templates, the application will draw rectangles around them. (Figure 16 and 17).

5 Generate a bird's eye view of a lane

5.1 Selecting Images

To develop and test this application 4 images (Figure 18) were used with different sizes, formats, and including different objects which could influence final results such as cars, bridges, road signs, and different road lane line colors.

5.2 Thresholding

Firstly, thresholding was applied to eliminate any parts of the image with different colors that differ from the normal colors of road lines, which are white and yellow. This was done by converting the color space to HSV, and then a range of possible values for the 'whites' and 'yellows' was used to create a mask to filter any color that fits any of those 2 ranges. The final result (Figure 19) shows exactly those colors in the images.

5.3 Edge Detection

Following that, there was a need to find edges, so we used the Canny Edge detection algorithm [3]. Before the algorithm was applied, the images were converted to grayscale, and a Gaussian filter was applied to reduce the amount of data to be processed and to reduce noise and, consequently, edge detection errors (Figure 20).

5.4 Region of Interest

After finding the edges, there was still information that was useless and could lead to bad results. To eliminate it, we defined a region of interest. Here we made one simplification assumption. Since the images have different sizes and points of view (POV), the region of interest is hardcoded after choosing the right points, which define a trapezoid that represents the road lane. Then the fillPoly [4] function of OpenCV did the rest, which was masking the outside points of the region with black so that it was ignored (Figure 21).

5.5 Hough Line Transform

The last step before applying the bird's-eye view was to detect and draw the lines of the original image. To achieve it, we applied the Hough Line Transform to detect lines (Figure 22). After that, there was still a problem: the result of the Hough Transform drew small lines and in some cases a lot of them. To fix it, we needed to turn the multiple lines into only one, so according to the slope of the lines, we could divide them into left or right lines of the road. Then, for each of them, left and right, we calculated a mean slope that would turn into the final line. After completing the previous stage, it was only left to fix the line length. This was done by finding the intersection points of both left and right. Then we applied Bresenham's line algorithm [1], whose aim is to locate every intermediate point needed to draw line AB on the pixelated computer screen so that the dotted line could be drawn instead of a straight line, as seen in the lab work specification. The final result is two dotted lines matching the respective road lines (Figure 23).

5.6 Bird's Eye View

To conclude the process, we implemented the bird's-eye view. Again, as seen in the definition of the region of interest, we measure the lane width in the closest pixel of the viewer and define a lane width for each of the images. This is going to be used to define the destination points to apply the perspective transformation. And to define the source points, we used the same array as the region of interest, the trapezoid that contains the road. The final result shows the road with highlighted lines and the bird's point of view (Figure 24).

5.7 Conclusions

Overall, the primary objective and concepts were carried out satisfactorily. Finding a better approach to specify the area of interest and the lane width and, as a result, improving the bird's perspective, is one of the application's enhancements that can be implemented.

References

[1] *Bresenham Line algorithm*. [Accessed Apr. 1, 2023]. 2023. URL: https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm.

[2] *OpenCV BRISK*. [Accessed Apr. 1, 2023]. 2023. URL: https://docs.opencv.org/3.4/de/dbf/classcv_1_1BRISK.html.

[3] *OpenCV Canny Edge Detection*. [Accessed Apr. 1, 2023]. 2023. URL: https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html.

[4] *OpenCV fillPoly*. [Accessed Apr. 1, 2023]. 2023. URL: https://docs.opencv.org/3.4/d6/d6e/group__imgproc__draw.html#gaf30888828337aa4c6b56782b5dfbd4b7.

[5] *OpenCV KAZE*. [Accessed Apr. 1, 2023]. 2023. URL: https://docs.opencv.org/4.x/d3/d61/classcv_1_1KAZE.html.

[6] *OpenCV knnMatch*. [Accessed Apr. 1, 2023]. 2023. URL: https://docs.opencv.org/3.4/db/d39/classcv_1_1DescriptorMatcher.html#a378f35c9b1a5dfa4022839a45cdf0e89.

[7] *OpenCV ORB*. [Accessed Apr. 1, 2023]. 2023. URL: https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html.

[8] *OpenCV SIFT*. [Accessed Apr. 1, 2023]. 2023. URL: https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html.

Annexes

Task 1

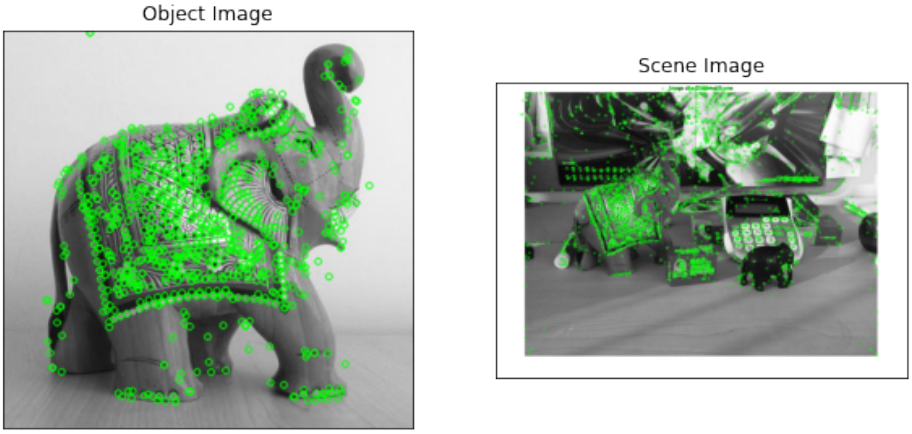

Figure 1: KeyPoint Detection using SIFT

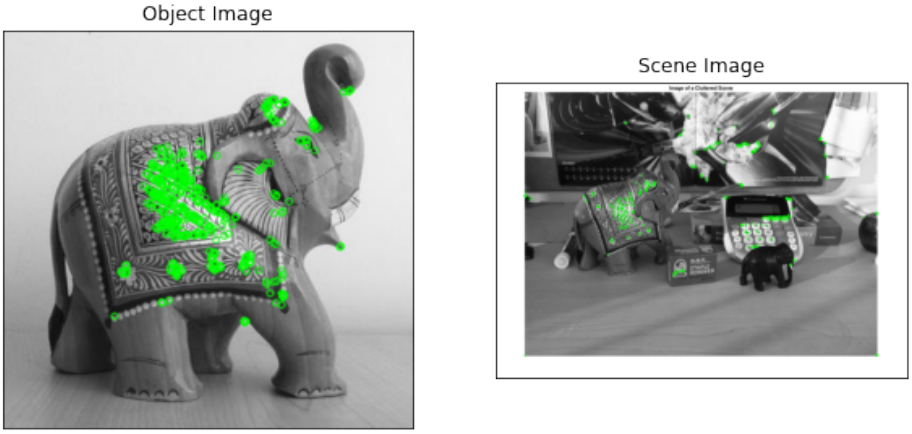

Figure 2: KeyPoint Detection using ORB

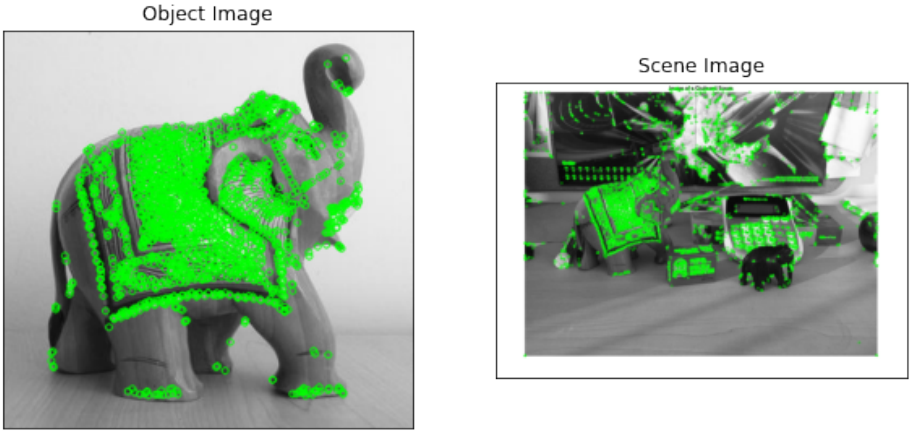

Figure 3: KeyPoint Detection using BRISK

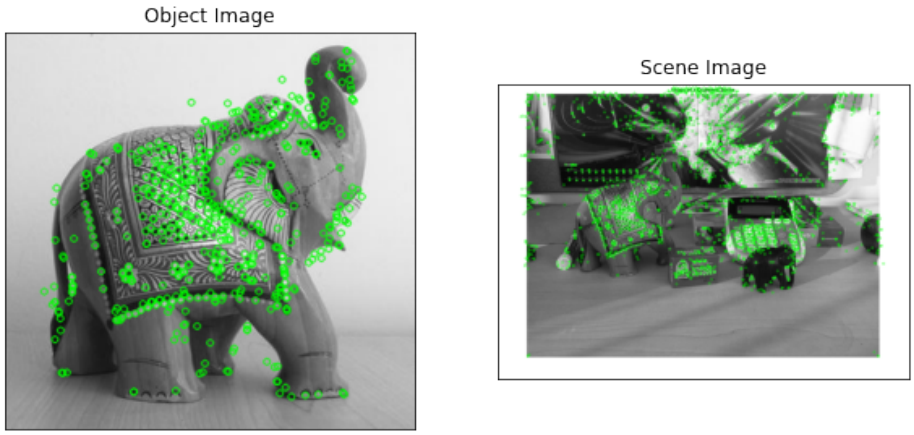

Figure 4: KeyPoint Detection using KAZE

Figure 5: Lowe Ratio Test with ratio 0.9

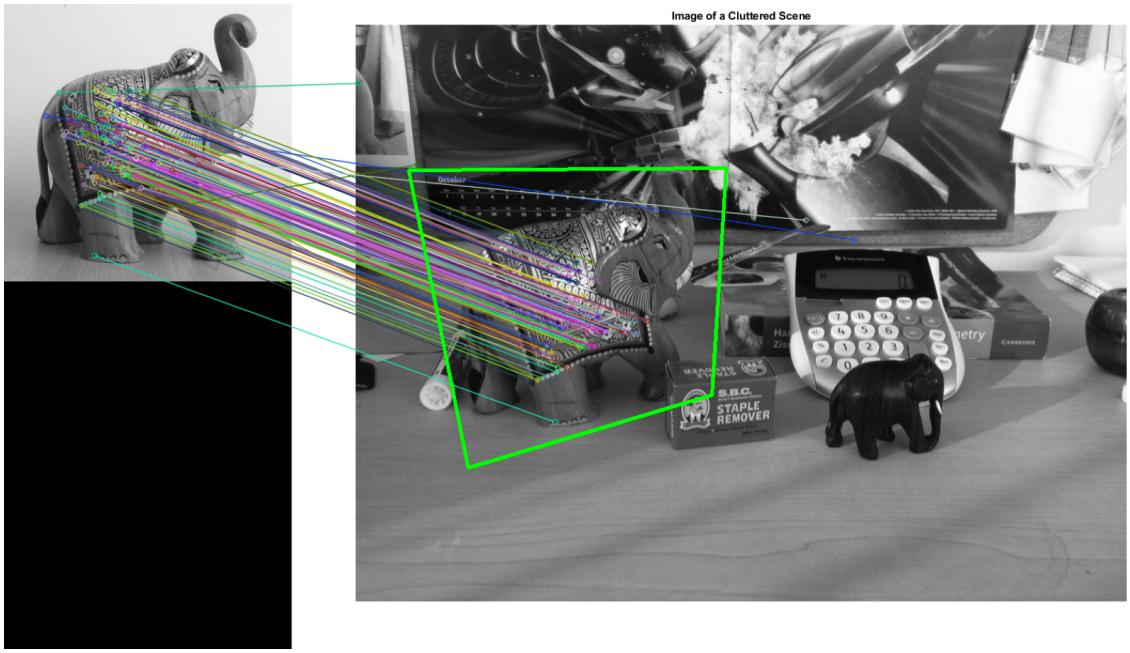

Figure 6: Lowe Ratio Test with ratio 0.7

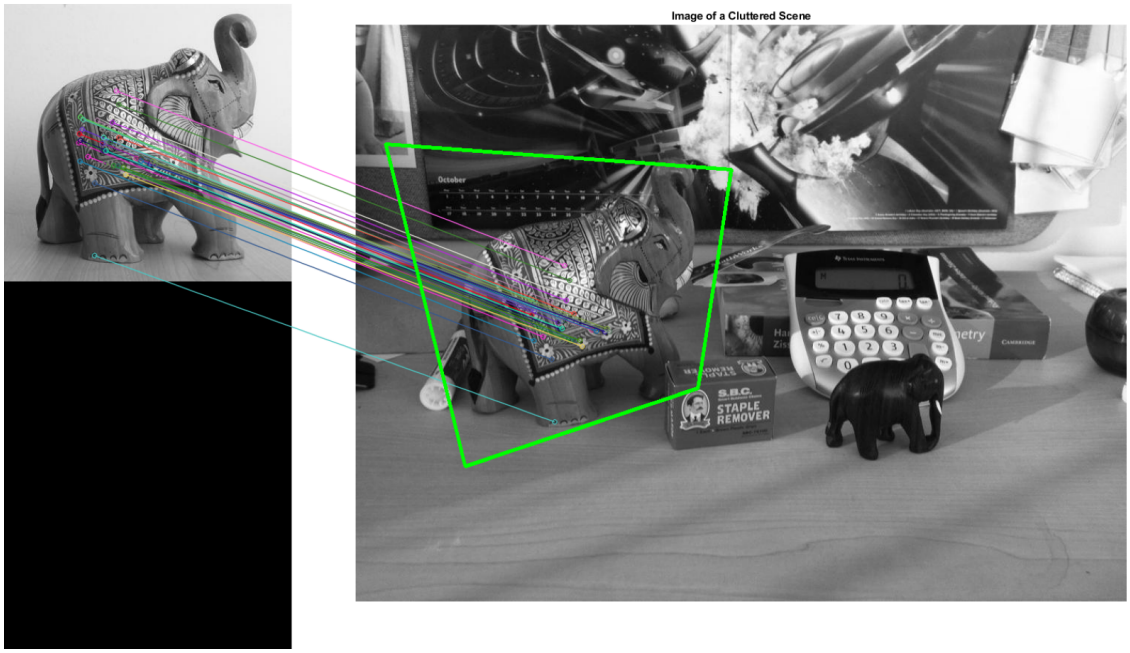

Figure 7: Lowe Ratio Test with ratio 0.5

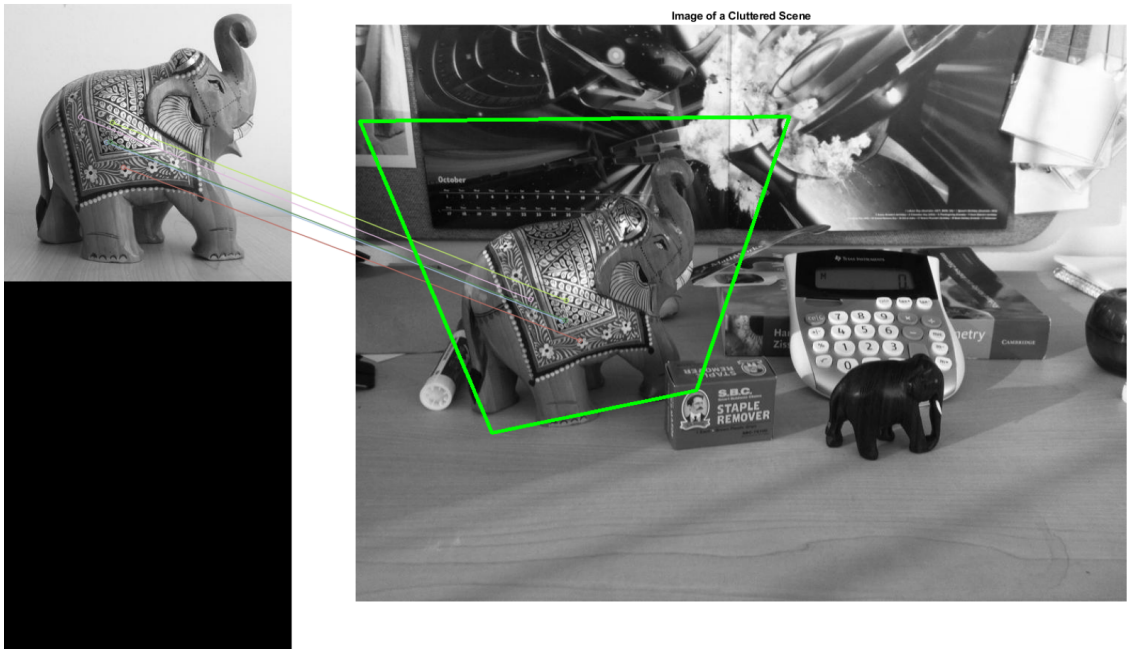

Figure 8: Lowe Ratio Test with ratio 0.38

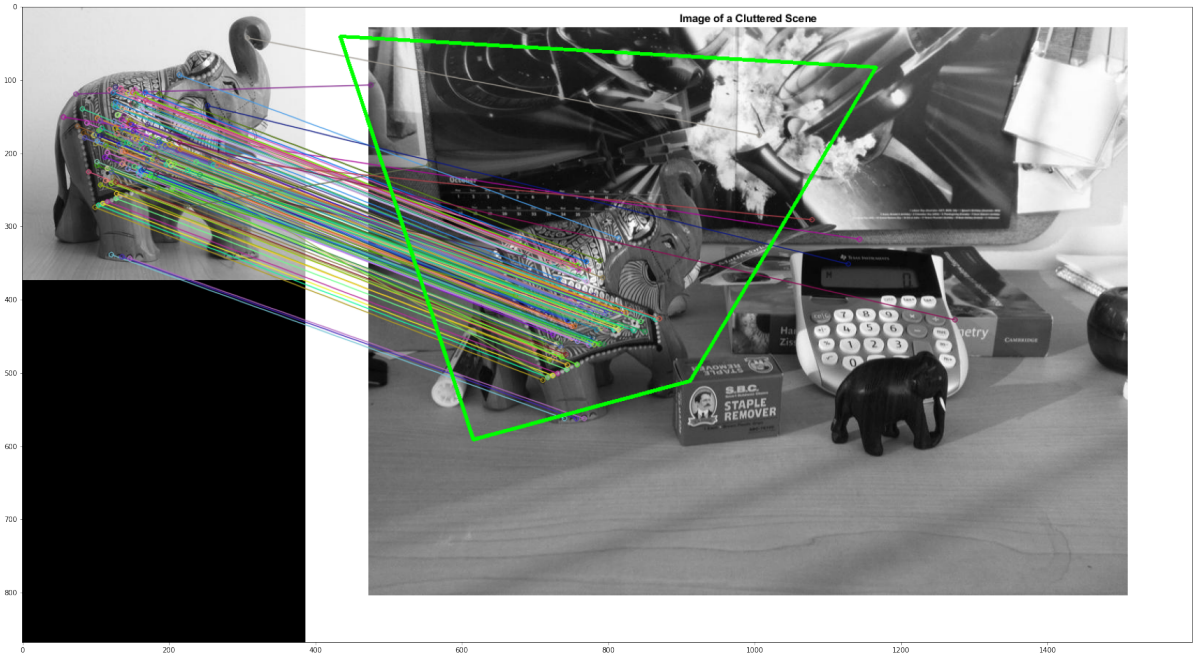

Figure 9: Find Homography without RANSAC

Task 2

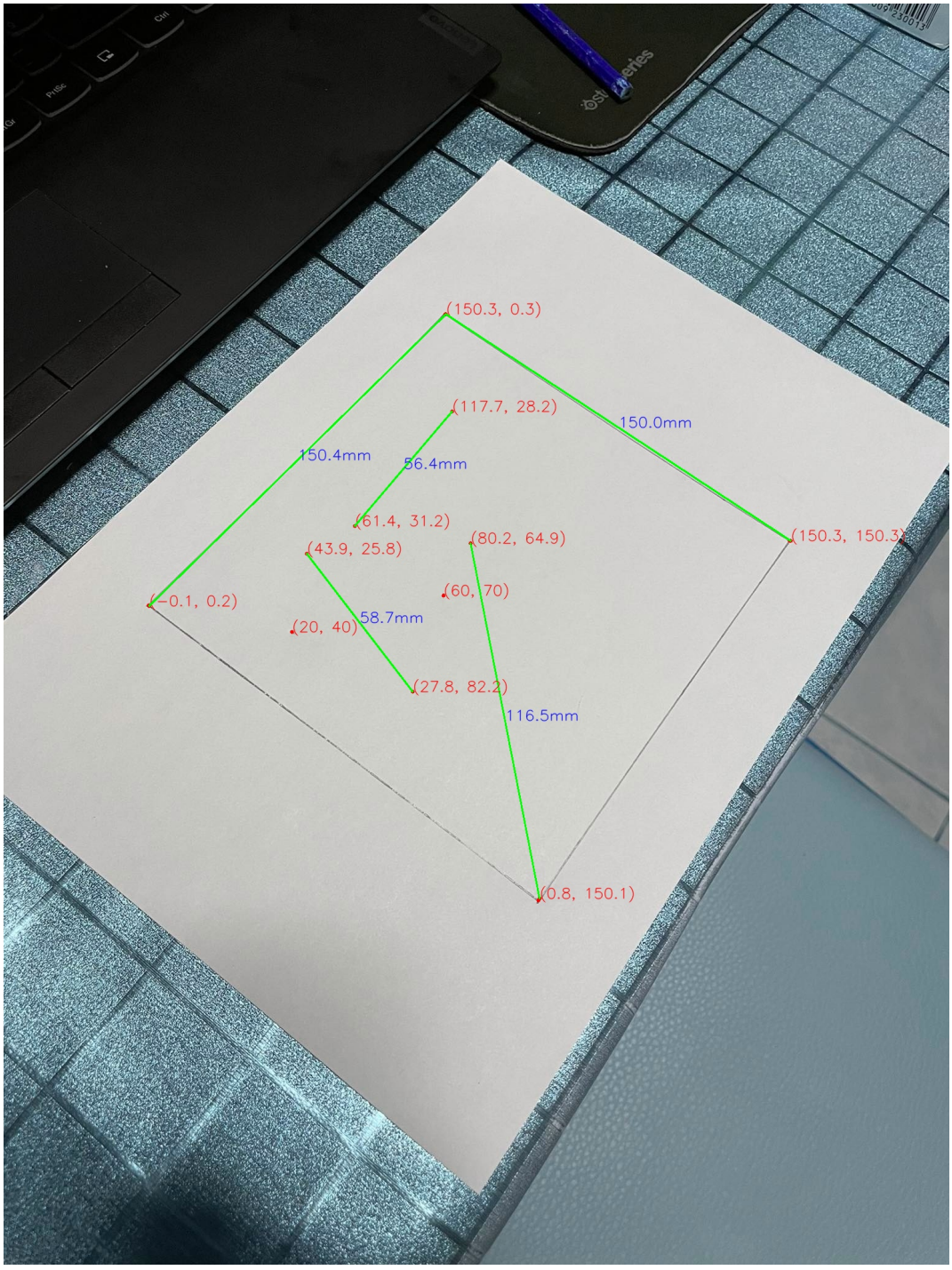

Figure 10: Measure distances with RANSAC

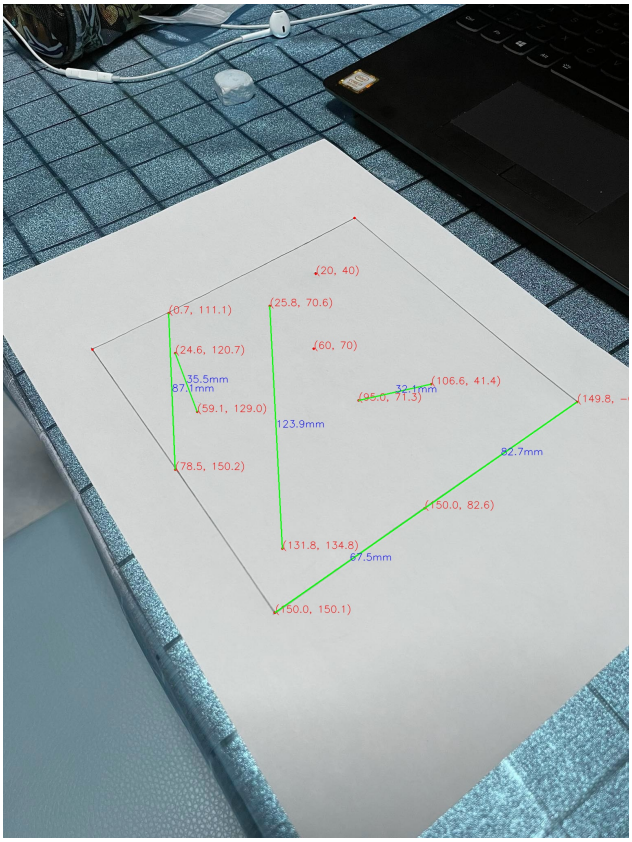

Figure 11: Measure distances with RANSAC

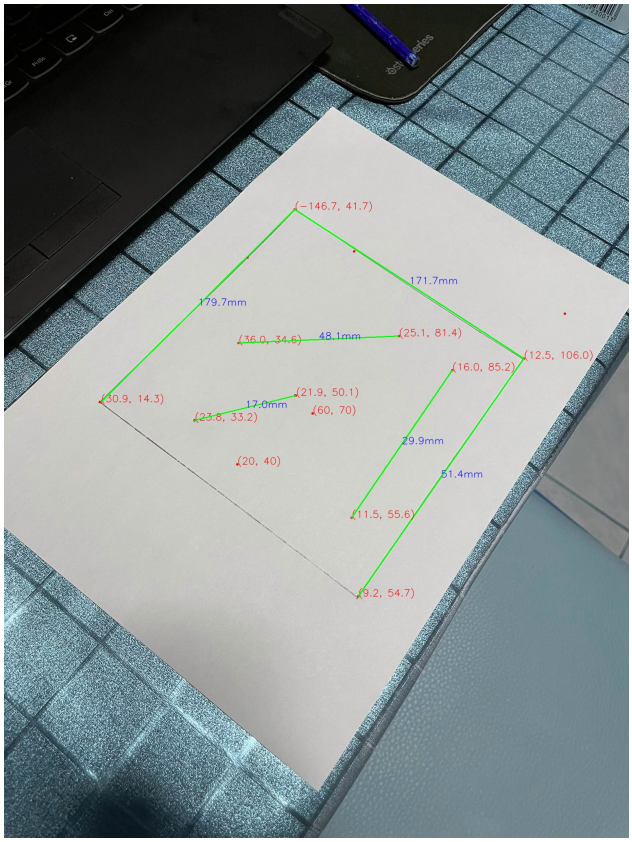

Figure 12: Measure distances without RANSAC

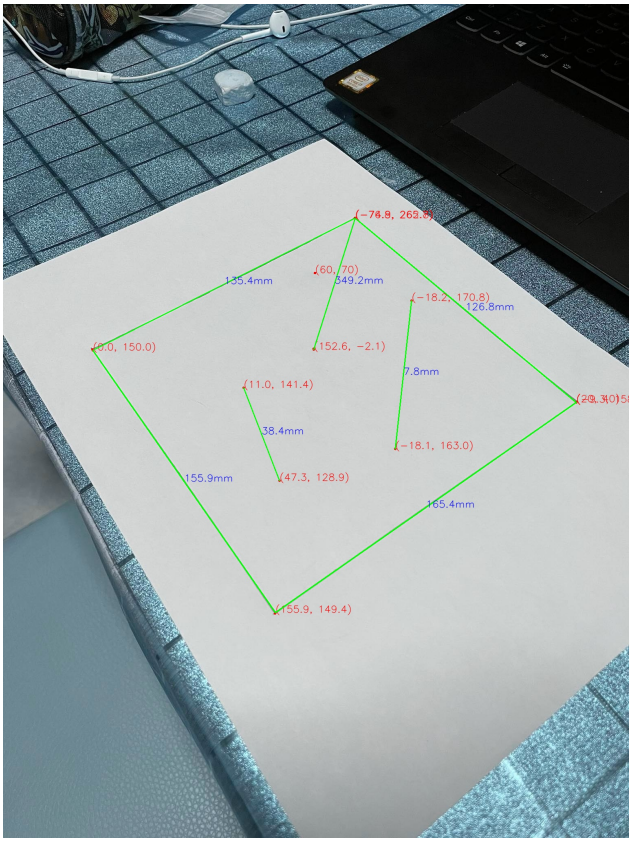

Figure 13: Measure distances without RANSAC

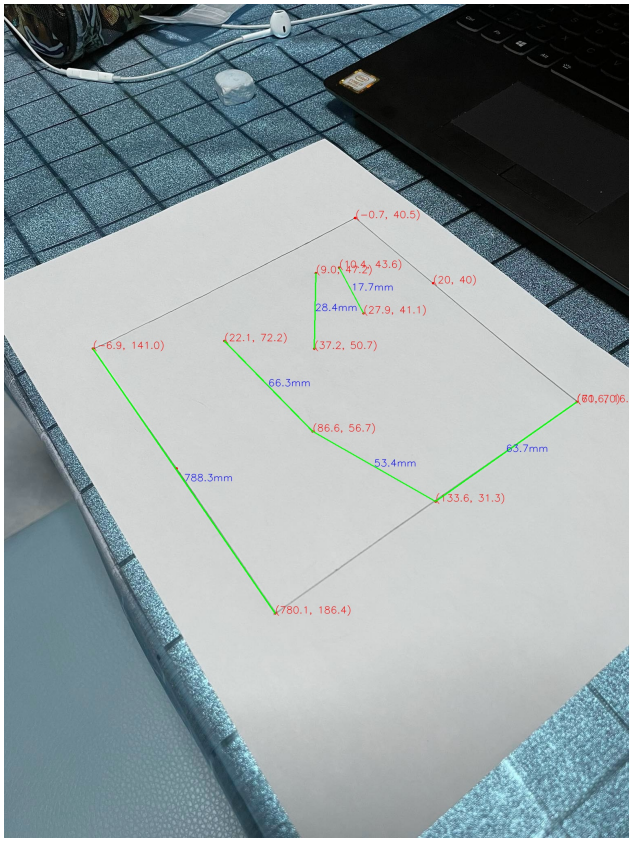

Figure 14: Measure distances without RANSAC

Task 3

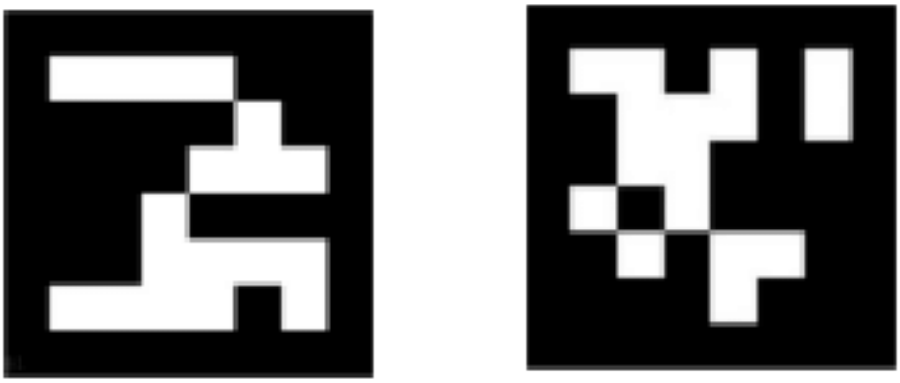

Figure 15: Template markers

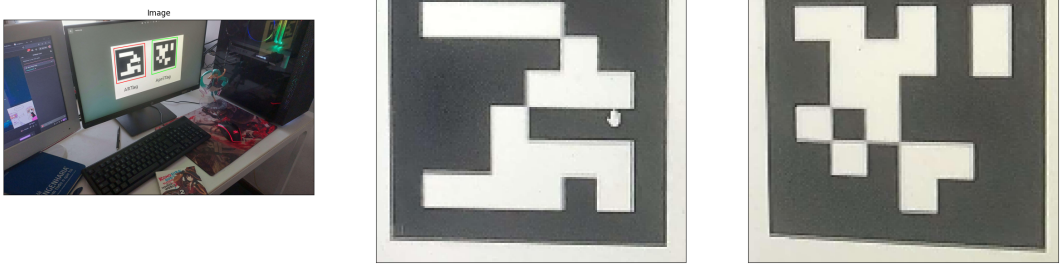

Figure 16: Perspective View

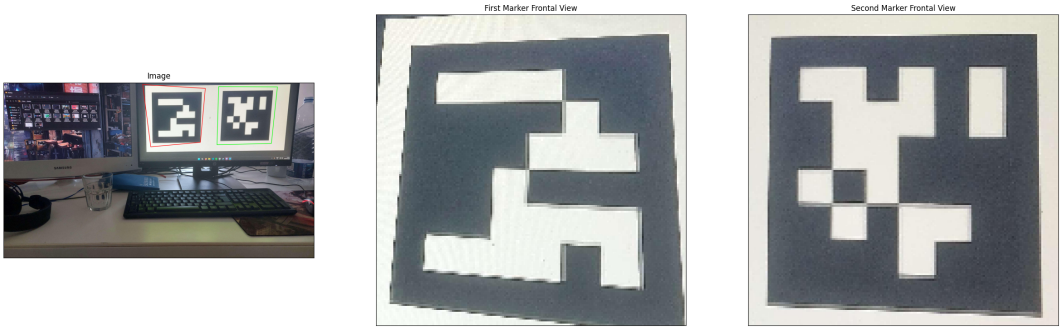

Figure 17: Frontal View

Task 4

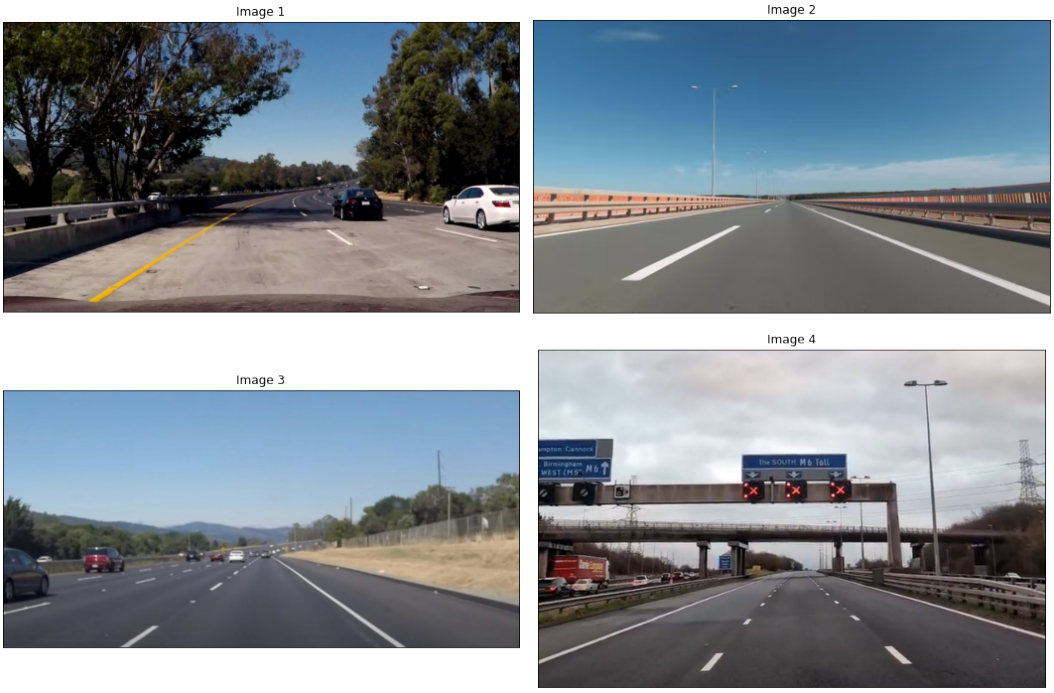

Figure 18: Images of 4 Road Lanes

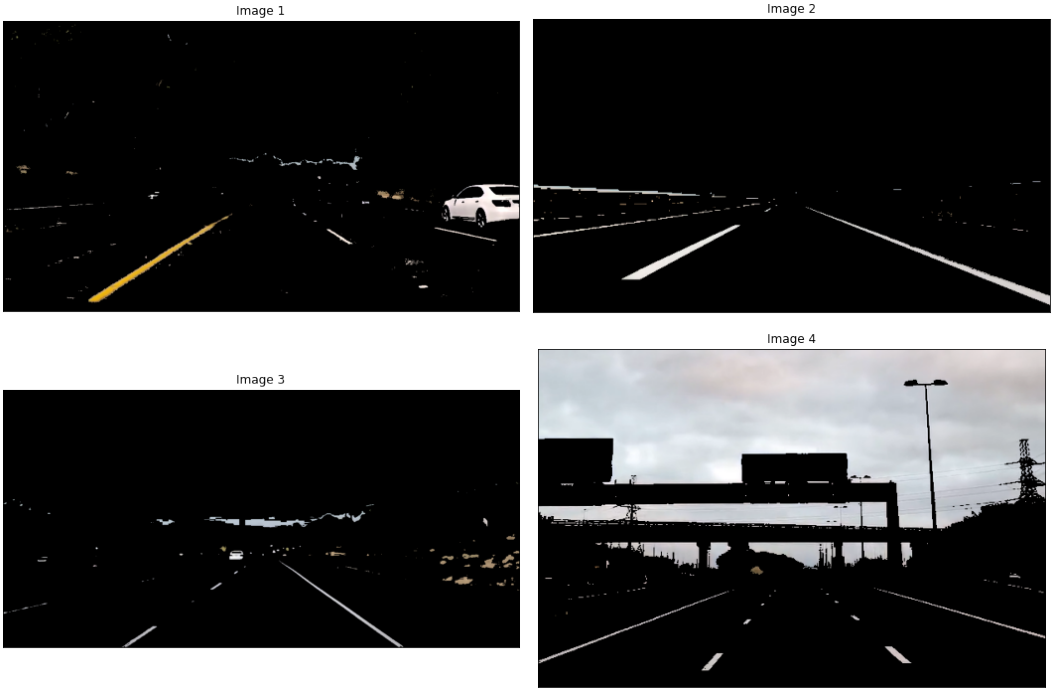

Figure 19: Threshold Applied to Road Images

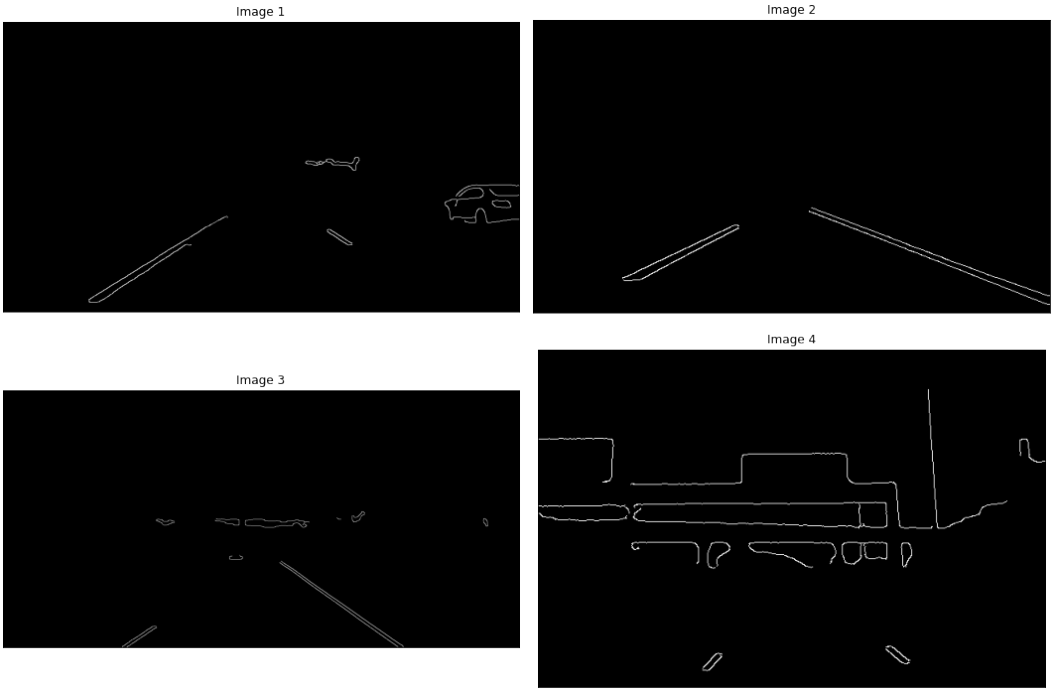

Figure 20: Edge Detection Algorithm Applied to Road Images

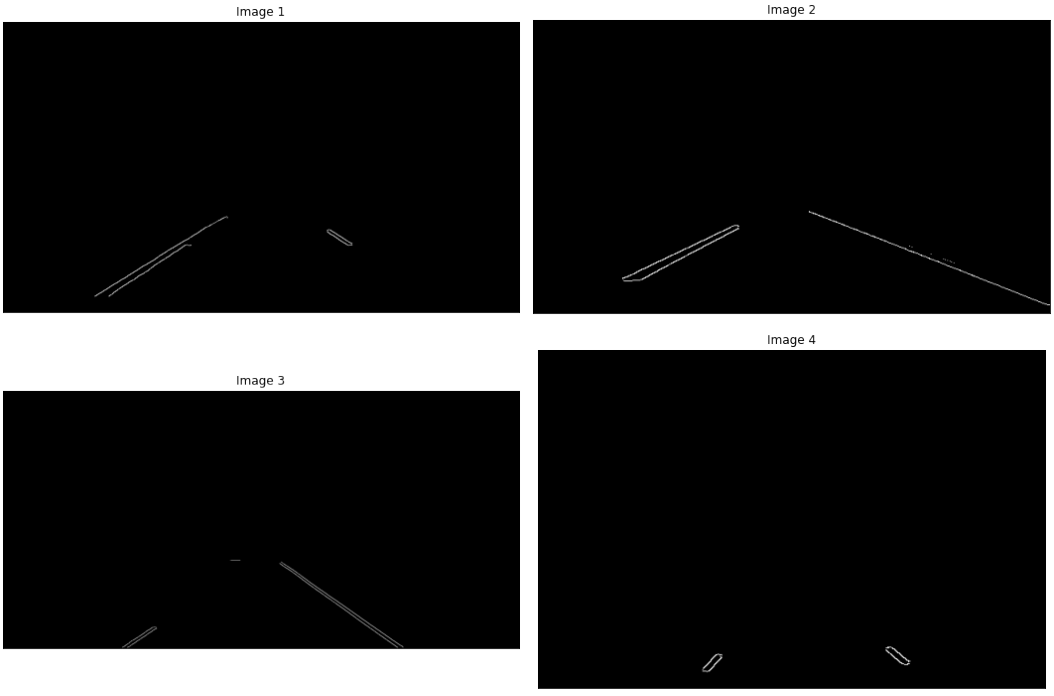

Figure 21: Select Region of Interest

Figure 22: Hough Transform Applied to Road Images

Figure 23: Fix Result Lines from Hough Transform

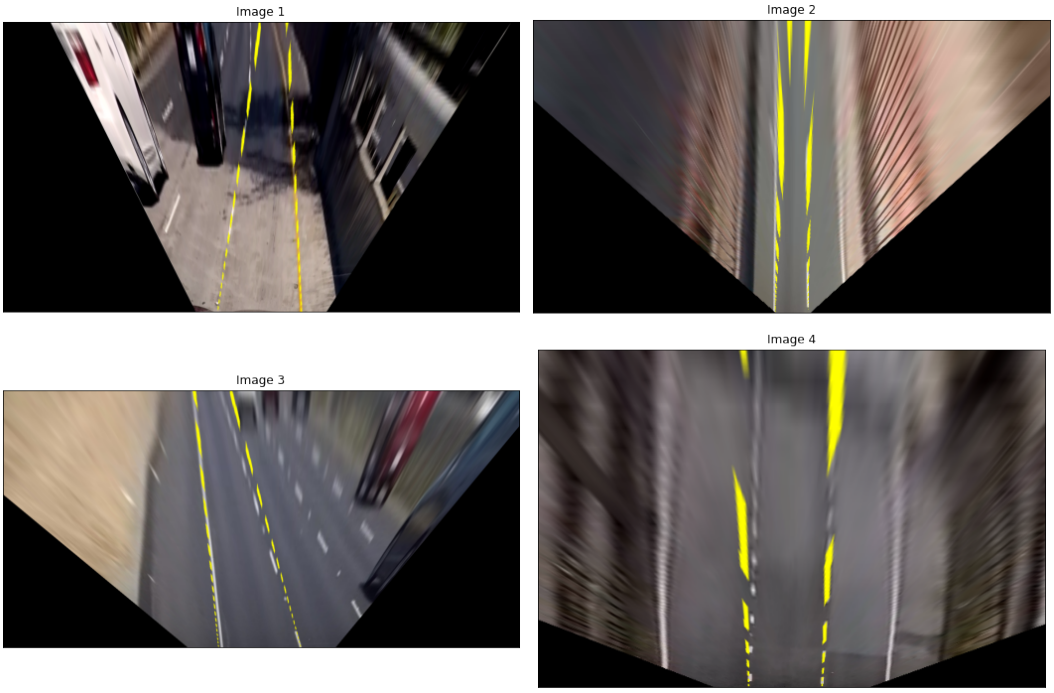

Figure 24: Bird View of Road Images