Homographies for Computer Vision

André Lino dos Santos
up201907879@edu.fe.up.pt
M.EIC
Faculty of Engineering of the University of Porto
Porto, Portugal

João Afonso M. D. Andrade
up201905589@edu.fe.up.pt
M.EIC
Faculty of Engineering of the University of Porto
Porto, Portugal

Maria José V. S. Carneiro
up201907726@edu.fe.up.pt
M.EIC
Faculty of Engineering of the University of Porto
Porto, Portugal

Abstract

In this report, we will explore some of the practical applications of computer vision techniques using the OpenCV library. Among these techniques we will primarily focus on the use of edge detection, segmentation, feature point detection and matching techniques, and geometric transformations to solve real-world problems in computer vision.
We will then describe our implementation of the techniques including some of the specific algorithms and parameters tested in each of the techniques.

We also present several applications of computer vision that use the techniques mentioned before, including localizing known objects, measuring distances on a plane with a single camera, recognizing objects using template matching, and generating a bird's eye view of a lane. Regarding these applications we describe the specific computer vision techniques used and the results achieved, including any limitations or challenges we encountered.

Overall, this report demonstrates the power and versatility of computer vision techniques and the OpenCV library for solving real-world problems, as well as the importance of understanding the underlying theory and parameters involved in these techniques.

1 Introduction

Computer vision has rapidly emerged as a key technology in modern computing, enabling machines to interpret and understand visual information from the world around us. From self-driving cars to augmented reality applications, computer vision is transforming the way we interact with technology and the world.

In this report, we explore the practical applications of computer vision techniques using the OpenCV library. OpenCV is a popular open-source computer vision library that provides a wide range of tools and algorithms for image and video processing. By leveraging the power of OpenCV, we can develop sophisticated computer vision applications that analyze, extract, and interpret visual information in real-time.

2 Task 1 - Localize objects

Develop an application that localizes an object in a scene, even when the object is partially occluded, using multiple feature point detectors and different parameters for matching the points and also with different objects and scenes and non-planar objects.

2.1 Implementation

2.1.1 Image preprocessing

In order to correctly use and implement the feature point detectors we first converted the images to grayscale

2.1.2 Feature Point Detectors

In order to detect and describe the key points in the image we utilized some of the OpenCv available detectors

- **SIFT (Scale-Invariant Feature Transform):** Is invariant to scale, rotation, and affine distortion and generates descriptors based on the gradient magnitude and orientation of the image around each keypoint.
- **Orb (Oriented FAST and Rotated BRIEF):** It combines the Oriented FAST corner detector with the Rotated BRIEF descriptor. It is also invariant to rotation and scale changes.
- **BRISK (Binary Robust Invariant Scalable Keypoints):** This is a feature point detector and descriptor that uses a scale-space pyramid to detect keypoints at multiple scales. Its descriptors are obtained with a binary comparison of the intensity between pairs of pixels around each keypoint.

2.1.3 Feature Point Matchers

Now that we have detected and computed the feature points in both the object and scene images we need to match them between the two of them and for this we tried:

- **Flann-based matcher:** Is a fast approximate nearest neighbor (ANN) search algorithm that is commonly used for feature point matching. It is quite robust to noise and outliers and is quite efficient when dealing with a large dataset
- **Brute force matcher:** It performs an exhaustive search over all the features of two images to find the best matches based on the distance between their descriptors. It can be tuned with parameters and can perform cross-checking to reduce false matches

In order to filter these matches and obtain the best possible result we also used and tested different parameters such as the **ratio** in the **Lowe's ratio test** and the influence of the use of **RANSAC** or not.

2.1.4 Results

Example of the detection of a book being occluded by other books in the scene.

In Lowe's test for feature matching, the ratio test is used to determine whether a feature in the first image has a unique best match in the second image.

The ratio test involves comparing the separations between the feature in the first image and its two closest matches in the second image.

The match is deemed legitimate and the feature is thought to have a unique best match in the second image if the ratio of the distances is below a predetermined threshold, which is often 0.8 or 0.7. The match is deemed ambiguous and is eliminated if the ratio exceeds the threshold.
As we can see in the pictures below the higher the ratio, the less matches occur but with higher confidence, resulting in a higher accuracy.

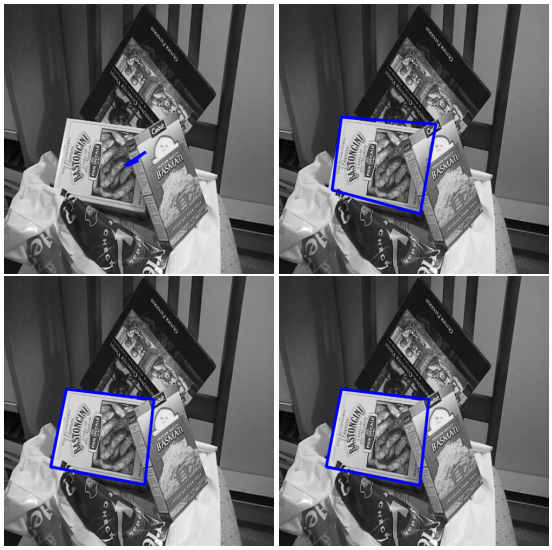

Figure 1: Using SIFT with the Flann matcher, RANSAC and Lowe's ratio test for ratio's of: 0.5, 0.6, 0.7 and 0.8 correspondly

As we can see in the pictures below the use of RANSAC highly influences the accuracy of the results which can be explained by its able to identify and discard outliers, allowing the algorithm to focus on the most reliable and informative data points. This makes it an effective technique for detecting objects that may be partially or completely occluded in a scene

Figure 2: Using SIFT with the Flann matcher and Lowe's ratio test. The image on the left uses RANSAC and the one on the right not

SIFT is known for its robustness against occlusion and scale variation, which makes it a good choice for object detection in cluttered scenes.

Although they have significant drawbacks, ORBand BRISK are other excellent options for object detection. Although ORB is quick and noise-resistant, it might not be as effective as SIFT at spotting keypoints in chaotic situations. Although BRISK is quicker than SIFT and offers scale-invariant feature recognition, it could not be as resistant to occlusion and scale fluctuations as SIFT.

In conclusion, the specific application as well as the properties of the sceneries and objects being detected affect the choice of feature point detector and descriptor. When dealing with occlusion and scale fluctuations, SIFT is a good option, although ORB and BRISK may be more appropriate in other situations where speed and robustness are important.

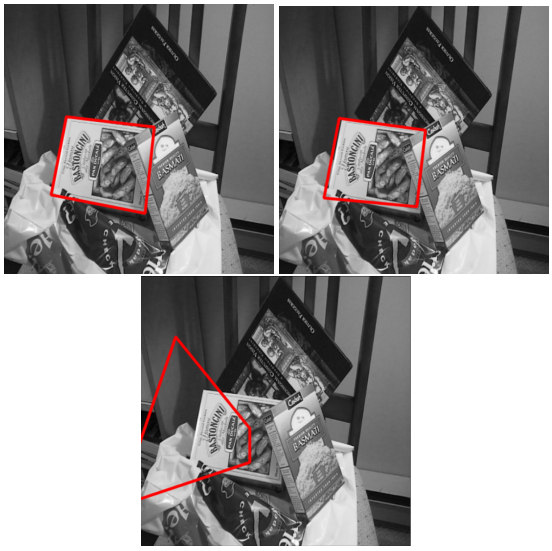

Figure 3: Detection of an occluded object with SIFT, ORB and BRISK in the same conditions

Figure 4: Detection of non-plannar objects using SIFT

3 Task 2 - Measure positions and distances on a plane with a single camera

In this task, the main goal is to develop an application that allows the measurement of the positions of points and distances on a plane with a single camera.

3.1 Pre Implementation

In order to implement and fulfill this task we first had to draw a square with some points on a white plane such as regular sheet of paper.

These points, which we need to know the real-life coordinates of, will then be used in the implementation of the algorithms in order to compute the homography that will correlate the points in the real world to the image. These points, which we need to know the real-life coordinates of, will then be used in the implementation of the algorithms in order to compute the homography that will correlate the points in the real world to the image.
These points are also measured in mm and we made the assumption that the origin of the referential was the top left corner of the square (0,0).

3.2 Implementation

In order to compute an homography we need at least 4 known points in the image and in the real life image

3.2.1 Selection of points

The selection of the points in the image was done manually and was then connected with each of the known point coordinates in the real life plane.

3.2.2 Homography Computation

Now that we successfully connected the points in the image to the real life plane we computed the homography.
On the computation of the homography we implemented two different solutions in order to understand whether or not the use of RANSAC in the homography had any impact in the overall results.

To analyse the impacts of RANSAC we manually established some wrong correspondences between the real life points on the plane and the points in the image. This was done in order to check whether or not RANSAC detects the presence of outliers and noise.

3.2.3 Without RANSAC

Without RANSAC, all the points are used to estimate the homography matrix. This method did not detect the presence of the wrong correspondences between points, achieving an accuracy of 1.
This means that the estimate will be influenced by any incorrect correspondences in the data (such as the one's we introduced)

3.2.4 With RANSAC

In contrast, RANSAC attempts to identify and remove incorrect correspondences, which can result in a more robust estimate of the homography matrix.
We tested the homographies with 2 wrong correspondences among 10 known points and RANSAC achieved an accuracy of 0.6. This way we can confirm that it removed the incorrect correspondences.

3.2.5 Distance and coordinates of any chosen points

After the computation of the homography we can now chose any two points in the image and apply the homography to them. After applying it we can then calculate the real life coordinates of the two points using:

$$cv2.perspectiveTransform()$$

This function is used to apply a perspective transformation to an image or a set of points. In this case it takes the homography matrix and applies it to the chosen image points.

Now that we know the real life coordinates of the two points we calculated the distance between them using a simple cv2 function that calculates the Euclidean norm (also known as the L2 norm) of a vector:

$$np.linalg.norm()$$

3.2.6 Results

As we can see in the picture below the points were sucessfully calculated as well as the distance between them.

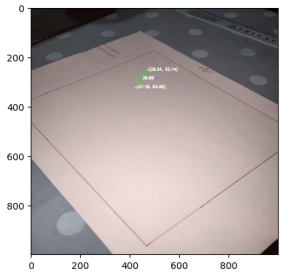

Figure 5: Points and Distance between points calculated with RANSAC

Now when it comes to the comparison between using RANSAC and not using it we saw obvious differences as we can see in the data below. When using RANSAC all the noise and outlier was ignored which resulted in successful calculation of the real life coordinates of the points. When we didn't use it the coordinates were not quite accurate and were off by a little bit because the noise was not ignored.

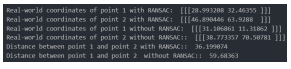

Figure 6: Points and Distance between points calculated with RANSAC and without RANSAC

4 Task 3 - Recognize planar markers using template matching

In this task, the main goal is developing an application that recognizes 2 different markers in an image caught from a perspective point of view. The app must draw squares around the markers if they match the pretended template.

4.1 Corner Detection

In order to detect the markers, a corner detection must be used. Considering their shape is a square, the application needs to find the 4 corners of the polygon and use them to calculate the homography and then compare it with the template. There are many algorithms to detect corners, so some of these were not tested like SUSAN(Smallest Univalue Segment Assimilating Nucleus) or FAST(Features from Accelerated Segment Test).

Considering that, the focus will be shifted towards the **Harris and Shi-Tomasi** regular corner detection methods, and since in this case they imply a big time complexity, a **contour** finder method will also be tested to provide a solution to the problem.

In the end, all processed corners were painted with a pure blue value in an image to ease their identification.

4.1.1 Regular Methods

Harris

The Harris corner detection method calculates the 'strength of a corner'. The first step is to calculate the gradient in order to parse the intensity variations within the image. It should be high on orthogonal regions. The C matrix is computed for the window surrounding each pixel and the likelihood of a pixel being part of a corner is able to be asserted based on the eigenvalues(cornerness score). Afterwards the points with large score can be evaluated and the weaker ones are suppressed(Non-maximum suppression). The low-confidence candidates are discarded according to the threshold, and the stronger ones remain.

Shi-Tomasi

The Shi-Tomasi algorithm is pretty similar to the Harris corner detector. The first part is exactly the same, as the gradient is calculated to evaluate the magnitude and direction of the intensity changes in the gray level image. Afterwards, things change as the corner quality measurement is different. Contrary, to the previous described algorithm, Shi-Tomasi pretends to find the minimum eigenvalue when assigning the corner score. Then, the same threshold and weaker corner candidates suppression paradigm is used.

Parameters

Harris corner detector has a specific OpenCV function, however, since the 'Good Features To Track', contains both the Shi-Tomasi and the Harris detectors, defined by a flag, it was the ultimate choice.

Given the number of corners outputed by these methods, a limit of 50 corners was established to limit the time usage to a 12 minute maximum. The quality level is a measure who defines the acceptance ratio for a corner to be accepted. A high value would be very peaky and leave the positive cases aside. The 0.01 minimum would provide an excessive number of corners. 0.05 was the used value. The minimum distance parameter was defined to avoid more than 1 corner to be marked in the same spot(a few pixels to the side). The distance defined was 10 pixels. The block size parameter refers to the size of the neighborhood around the pixels that is used to evaluate the corner likelihood on the spot. A larger block size measures the image globally, while a smaller one is more local.

4.1.2 Contour Finder Method

The Countour finder comes up as a solution to identify the markers, because it greatly reduces the number of corners detected, which reduces time usage by a lot.

The find contours OpenCV function takes a binary image as input. In order to catch the template in the images, an inverted threshold was used. Then, the retrieval mode is specified, in which only the external contours were used. The approximation method was the simple chain approximation.

The algorithm scans the image for pixel connections and groups them into contour segments. It is based on the method described in Suzuki and Abe's algorithm described in the paper Topological structural analysis of digitized binary images by border following. The retrieved data are then the contours according represented in sequences of points. [7]

Since the goal is to find the squared marker, the contour may then be processed to be approximated using an approximation to polygon technique. This allows the corner identification with ease. Using the Douglas-Peucker, incorporated in an OpenCV function, the accuracy of the approximation is high and can be considered a much more efficient method for this specific case. However, in a situation where many markers are in the image, we may end up with a high level of corners anyway. Despite that, we could still adapt our code to work only using this method and it would be easy to increase its efficiency drastically. However, we would lose the generalisation ability so we decided to maintain it usable to both algorithm types because of comparison reasons.

4.1.3 Algorithm Comparison

When comparing the 2 regular algorithms, there were not many differences. Efficiency-wise, it was basically the same. The number of detected corners is a small chunk higher in Shi-Tomasi(on the left), since for the same parameters it was a little bit more sensible than Harris(on the right).

Figure 7: Shi-Tomasi (Left) vs Harris (Right) corner detectors

Despite the similarities between the previous two, the find contours method differs highly. Since the first two provide a diverse set of corners from anywhere in the image, their number escalates rapidly. To find the four matching points. Considering a set of 50 points, 5527200(50A4) iterations will pass by. Obviously, this consumes a lot of time.

Considering the contour only matches the two squares, we get the exact points we need, and the time usage goes from a minute

scale to under-second ranges. Evidently, in the end, we preferred to utilise the contour method. The time consume is described in the following graph.

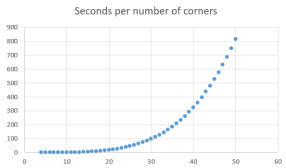

Figure 8: Seconds to match the template according to the number of corners detected

4.2 Vector similarity

Regarding the temporal processing issues described previously when a high number of corners is present, some measurements needed to be taken to make the solution feasible. The numbers seen on the graph containing the time usage per number of corners are already optimized. Before that, the template matching was from 5 to 10 times slower. The opportunity to accelerate the process was found using the metric of **vector similarity**.

This is asserted using two key parameters, the size and the angle between vectors. Since the templates are squares, even if they are seen from a perspective, the opposite sides, should be parallel, and identical in size. This means, we can discard a 4 point combination by creating 2 vectors and comparing them. Evidently, there are always systematic errors. Therefore, thresholds of 95% of the size and 11 degree were used to accept a possible deviance. Only vectors inside these ranges were then put to proof in template matching, a much more time consuming operation.

4.3 Template Matching

Template matching, the core of the problem is actually relatively simple.

Homography

Firstly, an homography is defined using the previously iterated 4 points relatively to a predefined square sized view, equal to the template coordinate system. The matrix that maps the original corners to the new space is stored to apply the reverse later on. Since the different templates calculate different homographies, the original image may contain the two squares on different planes.

Warp Perspective

Then, in order to compare the image found template with the original, it needs to be transformed into the new space, using the previously calculated homography so that the pixels are matching and the possible similarities are maximised. This is done using the warp perspective OpenCV method.

Matching

The match template function slides through the provided image, and compares the overlapping its pixels against the template ones, storing their similarity ratio in a matrix. This can be computed using different methods. Using the minMaxLoc function, the best matches can be found as global minimums (TM_SQDIFF) or maximums (TM_CCORR or TM_CCOEFF). In this case, the mode used was TM_CCORR_NORMED, which means the maximum is the value to search for.

To force a strict evaluation, a high threshold of 0.8 was defined. The best matches would usually fall within the 0.80 and 0.90 range, since it is really difficult to get a perfect match, when lighting and image quality problems are at stake. This value proved to always identify the templates, while also denying unwanted matches with high similarity,

Using the inverse of the Homography Matrix on the matching results, we can get the original corner positions and draw lines to mark the template border.

4.3.1 Others

Other helping specifications of the application were, for example, the itertools permutations functions, which allowed the iteration through all the corners without writing a nested for loop with depth 4.

When in the presence of excessive corners, every 10000 iteration a message is printed to keep track of the time and the process status.

4.4 Results

In sum, the main goal of using template matching to recognize two different planar markers, using corner detection, was completed with success. Even though the nature of the problem was leaned toward the perspective transformation and template matching, the main issue ended up being the corner detection part. Even though the regular corner detection algorithms are used broadly, in this case, it was not the most useful application for their capacities. Despite that, the contour finding method ended up coming up as a good solution and allowed the obstacle to be surpassed. [5]

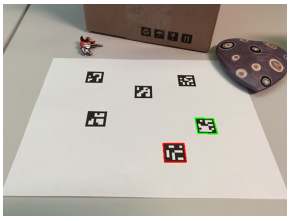

Figure 9: Matching templates being found in image with other possible similar templates

5 Task 4: Generate a bird's eye view of a lane

The main goal of this task is to detect lane edges on driver's view of a road, generating a bird's eye view of the lane. The pipeline followed was applied sequentially by the stages described in the next sections.

5.1 General Assumptions

In order to simplify the problem at hand, we made some assumptions that were maintained throughout the pipeline:

(1) Camera is mounted on the car and stays at the same location, pointing straight ahead.
(2) Lanes are flat, straight and can only be yellow or white.

(3) Highway scenario, with no occlusion in the ROI and good good weather conditions.

For testing purposes, two images (lane1 and lane2) were captured by the same camera and so have the same resolution and the last image (lane3) was considered to showcase an example were the road colors are not uniform.

No curved lanes were considered or any case in which the weather conditions weren't favorable. Shadows in the lane can also affect the results since they weren't taken into consideration. For both of those cases, CLAHE could be applied before the segmentation in order to better the image contrast.

5.2 Camera Calibration

Images gathered by moving vehicles are generally videos captured by dashboard cameras that may be prone to radial distortions which result in inconsistent magnification depending on the object's distance from the optical axis [3]. For that reason, in order to find the best match in lane markings, camera calibration should be applied to remove any distortion in the image caused by the camera collection. The distortion removal effects for lane2 and lane3 can be found on the appendix.

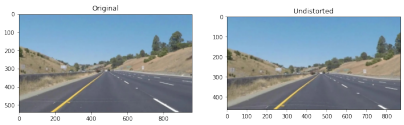

Figure 10: Results of distortion removal for lane1

Overall, the undistorted results were similar to the original images, so there was little to no distortion on the original cameras.

5.3 Color Segmentation

Taking in to account assumption 2, the lanes can be isolated according to their color in the image: yellow and white pixels are maintained and the others are thresholded to black. We tested with 6 different colorspaces (RGB, HSV, HLS, YUV, LUV, LAB) in order to choose which best detected the lanes. HLS was the colorspace that mostly highlighted the lanes over the other image components that were also yellow or white. As a later improvement, a color mask could be applied by gathering the best channels from different colorspaces. The results of the full colorspace analysis can be found on the appendix.

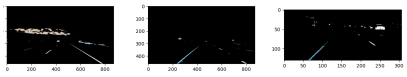

Figure 11: Results of color segmentation on HLS colorspace

5.4 Lane Line Detection

The lane line detection was applied to the color segmented images obtained from the previous step. We chose to use Canny Edge detection in order to find the lane lines present in each image. Firstly, we converted the color segmented images to grayscale, then we applied Gaussian blur to remove some noise present on the image. Afterwards, we applied the Canny Filter which selected the lines present in the image bellow. As observed, some of the background

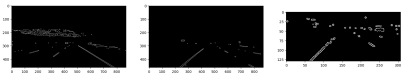

Figure 12: Results of Canny Edge Detection

was also detected alongside the lane lines, so we applied a Region of Interest (ROI) selection to the given images. As stated in section 5.1, lane1 and lane2 were captured in the same conditions and have the same resolution, so the ROI considered for those images was the same. As for lane3, we had to consider a different ROI considering the image dimensions. As a later improvement, the ROI could be automatically detected instead of manually selecting thresholds considering each images proportions. Afterwards, the lines were

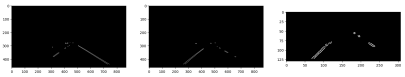

Figure 13: Results of ROI selection

isolated using the Hough Transform and were superimposed on the original images. Since the Hough algorithm produces many lines and only on the detected places, we had to condense the lines into a single trace and extrapolate them in order to cover the full lane line length.

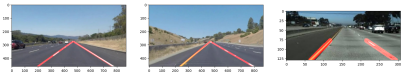

Figure 14: Results of Hough Transform, after condensing and extrapolating.

5.5 Perspective Transform

With all the lane lines detected, the final step would be to transform the image from a driver's perspective to a bird's eye view. We chose to keep the top unchanged while shrinking the bottom, in order to keep the most amount of information from the viewing angle. Just like in the ROI, the points selected were adapted to each image according to their resolution (same for lane1 and lane2, different for lane3).

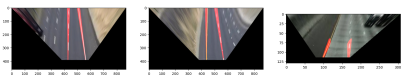

Figure 15: Results of distortion removal for lane3

Overall, the results were satisfactory and we could detect lane lines as well as generate a bird's eye view of the lane. Lane3 was the image with the worst performance, probably due to the fact that the lane was mismatched in color and, in perspective, it was harder to

distinguish the lines from the lighter, more worn out, asphalt color. In later improvements, we could choose to apply the perspective transform before the lane line detection, so that the lines are more clearly identified in all cases.

6 Conclusion

In conclusion, this report highlights the practical applications of computer vision techniques using the OpenCV library. Through the use of edge detection, segmentation, feature point detection and matching, and geometric transformations, we can solve real-world problems in computer vision. The report provides specific algorithms and parameters tested for each technique and presents several applications that use these techniques to achieve good results, such as localizing known objects, measuring distances on a plane, recognizing objects using template matching, and generating a bird's eye view of a lane. The report emphasizes the importance of understanding the underlying theory and parameters involved in these techniques for developing powerful and versatile computer vision applications. Overall, this report demonstrates the potential of computer vision to transform the way we interact with technology and the world.

References

[1] Mohamed Ameen. 2017. Lane Lines Detection Using Python and OpenCV. (2017). Lane Lines Detection Using Python and OpenCV, Last Access Date: 06/04/2023.

[2] Wong P.K. Yang ZX Chen, Y. 2021. A New Adaptive Region of Interest Extraction Method for Two-Lane Detection. Int.J Automot. Technol. (2021). A New Adaptive Region of Interest Extraction Method for Two-Lane Detection. Int.J Automot. Technol., Last Access Date: 06/04/2023.

[3] Moataz Elmasry. 2018. (2018). Computer Vision for Lane Finding, Last Access Date: 06/04/2023.

[4] Luiz Gonzalez. 2018. Traffic Lane Detection using Color Space Thresholding. (2018). Traffic Lane Detection using Color Space Thresholding, Last Access Date: 06/04/2023.

[5] OpenCV. 2023. (2023). Template Matching. Aruco Templates Example in OpenCV, Last Access Date: 06/04/2023.

[6] OpenCV. 2023. Camera calibration. (2023). Computer Vision for Lane Finding, Last Access Date: 06/04/2023.

[7] Satoshi Suzuki and Keiichi Abe. 1985. *Computer vision, Graphics, and Image Processing* 30 (1985), 32–46. Topological Structural Analysis of Digitized Binary Images by Border Following, Last Access Date: 06/04/2023.

[8] Luca Venturi. 2021. Lane detection with OpenCV — Part 2. (2021). Lane detection with OpenCV — Part 2, Last Access Date: 06/04/2023.

7 Appendix

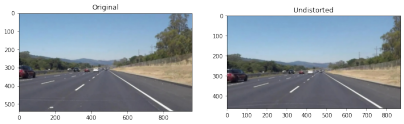

Figure 16: Results of distortion removal for lane1

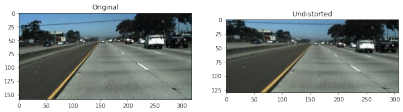

Figure 17: Results of distortion removal for lane3

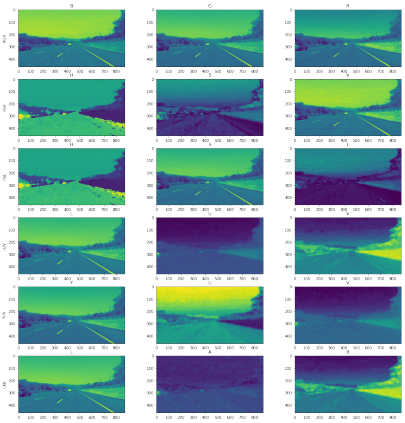

Figure 18: Different colorspace analysis

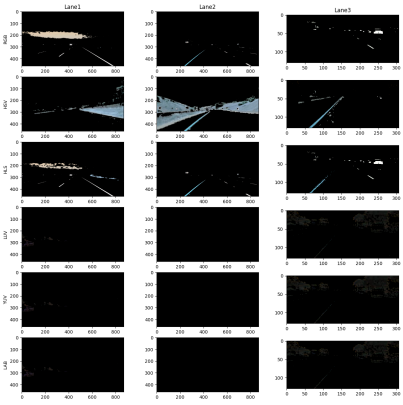

Figure 19: Full color segmentation to all colorspaces