Universität Siegen
Siegen, Germany.

On Bird Eye View Transformation and Lane Detection

A Report

Report by:

Shishir Pokhrel
Mechatronics Student

Contents

List of Figures

List of Tables

1 Implementation and Results

1.1 Experimental Setup

The development environment is Linux, and the programming language used in this project is Python. OpenCV library is extensively used throughout. The cameras used here are basic webcams from Logitech, which can be easily accessed by the computer. The test environment was set up with lane lines drawn over paper. The two cameras are set up on a cardboard box, taped for sturdiness and to restrict movement. The box is used as a toy car, to move over the lanes during testing.

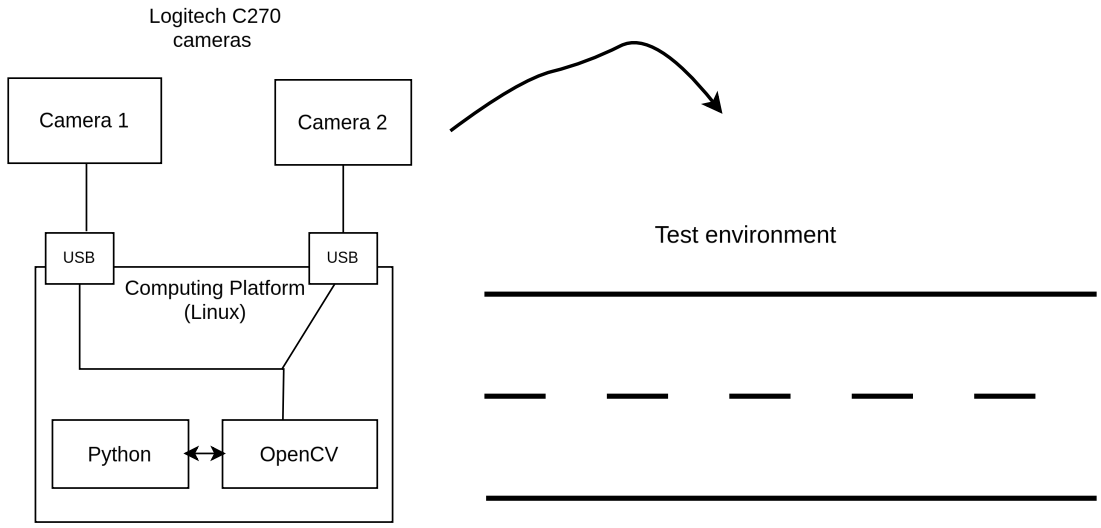

Figure 1.1: Experiment setup

Moreover, a calibration board (checkerboard pattern) is printed out and pasted on a hard board, which is used in the calibration steps. During testing phase, some random objects like a Die and Pebble (although irrelevant but useful) were used, which will appear in later parts of this project.

The following subsections will discuss the details of the use of this experimental setup and the results obtained.

1.2 Camera Calibration Results

The camera calibration process according to Chapter 3, results in the intrinsic matrices, along with distortion matrices for the individual cameras, and Rotation and translation parameters for the stereo camera. These values are given in this section. Normally, a pinhole camera is not supposed to have distortion, ideally. However, in this case, the calibration process did give some distortion.

$$C_{m1} = \begin{bmatrix} 1.49218452 \times 10^3 & 0 & 6.03490449 \times 10^2 \\ 0 & 1.49610277 \times 10^3 & 4.03113144 \times 10^2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0.14043345 & 0.02129057 & -0.02553297 & -0.01685553 & -1.2258668 \end{bmatrix}$$

$$C_{m2} = \begin{bmatrix} 1.51120008 \times 10^3 & 0 & 6.48536573 \times 10^2 \\ 0 & 1.51645403 \times 10^3 & 5.15486362 \times 10^2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0.08486609 & 0.10679398 & 0.01633373 & 0.00649924 & 0.11807069 \end{bmatrix}$$

Furthermore, although the same camera model was used, (albeit produced a few years apart), these cameras show intrinsic parameters deviating by around from each other slightly.

The rotation and translation for Stereo setup is given with the following :

$$R = \begin{bmatrix} 0.99835944 & -0.03553422 & 0.04489709 \\ 0.0430387 & 0.9828851 & -0.17912162 \\ -0.03776373 & 0.18076007 & 0.98280196 \end{bmatrix}.$$

$$T = \begin{bmatrix} -3.9336552 \\ -0.17502972 \\ 0.46588329 \end{bmatrix}$$

1.3 Bird Eye View

The methodology section described how the bird eye views for different camera poses were created. In this section, the results of the initial bird eye view of a book is shown both by geometric approach and by using homography matrix. The reason for using this book is that the letters ('TIGER') are visible in the BEV image, and thus highlight the transformation.

A geometric approach as conducted previously by Calligaris [1] is adapted and presented below. This approach uses a geometric relationship from a camera perspective to bird's eye view, iterates over the pixels and converts them with the relationships described in Chapter 2, and provides a bird's eye view. Figure 1.2 presents an example of this approach. The original image is the same as in Figure 1.3.

Figure 1.2: Bird Eye View using Geometry

Figure 1.3 shows the results of the operation of finding the chessboard corners and the warped perspective view. In a nutshell, the Test image, for instance, is transformed using the homography matrix H, into a bird's eye view obtained by evaluating the difference in the pose of the two chessboard images, which is essentially estimated using the corners of the chessboard pattern. The Bird's eye view image includes about half of the height of the original test image. This is an important aspect because the change in homography extends the scene in the Birds Eye view.

The homography matrix obtained is

$$H = \begin{bmatrix} 2.891376 & 4.748639 & -1220.122837 \\ -0.333787 & 13.028430 & -4695.507091 \\ -0.000284 & 0.007148 & 1.000000 \end{bmatrix}$$

The observations from these two methods are summarised in Table 1.1.

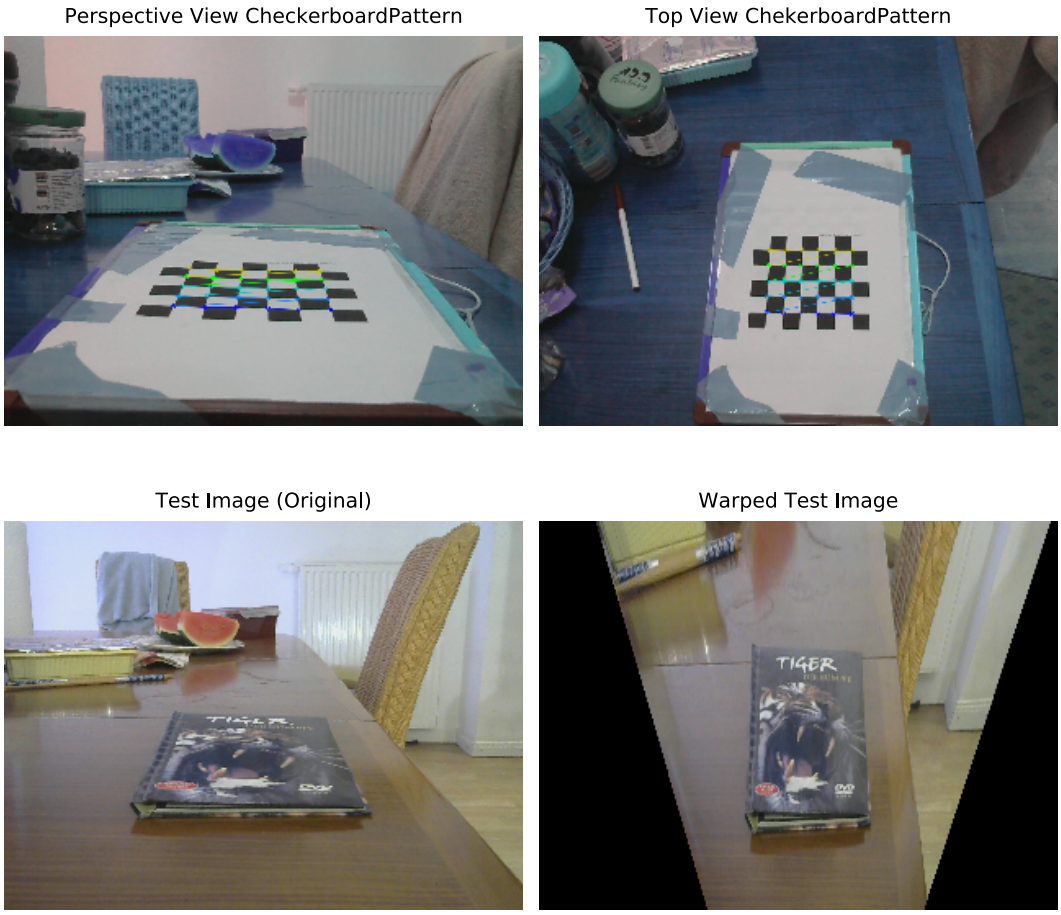

Figure 1.3: Steps in obtaining the Bird Eye View.

The bird eye views presented use Homography from a stationary camera. In other words, this homography matrix gives the BEV as the basis. The next step is to revise this BEV transformation if there are changes in the camera pose. This is done by using the IMU sensor to gather inertial measurement data, to feed to a rotation matrix that further transforms our homography H.

From Chapter 2, we know that for planar homography, the changes in the vertical or Z axis is set to 0, effectively making the homography matrix a 3x3 instead of a 3x4 matrix. Furthermore, the bird's eye view changes for a change in camera pose is relevant for rotation about the planar axis, either for rolling or for pitching. In the case of yaw, the bird's eye view is the same, as the ground plane is not changed. Therefore in this implementation, the two dominant angles, roll and pitch are used. In Figure 1.4 the roll, pitch and combined rotation of a vehicle, and the bird eye view is presented. The horizontal and vertical lines are markers to visually help assess the changes in BEV.

Method	Pros	Cons
Geometric	The camera angle and height are directly given as input which is used to compute the transformation, meaning that this method could be generally suitable for dynamic scenes.	The counter-argument for this method is that it uses focal length as a parameter, and therefore requires precise focal length calibration, as camera focal lengths can have tolerance errors. This method also has a high computational complexity and load. The runtime for obtaining the BEV image in Figure 1.2 was 14 seconds in a 1.8GHz 4-core processor for instance.
Homography-Based	This method is faster and less complex, in use case therefore it could be used on low-power devices like Raspberry pi. It is possible with this method to obtain a bird eye view on the video feed, and present the BEV as a video in the same rate as the original.	However, this method requires accurate homography estimation as the calibration process has the sensitivity to errors. Since the homography matrix is computed with pixel coordinates (due to the estimation of corner points (see Figure 1.3), any use of external devices must be calibrated for this.

Table 1.1: Comparison of Bird's Eye View Methods

The BEV images for Roll angle of -10 degrees and pitch angle of 10 degrees show the extension of the scene in the BEV image (compared to 0 degree rotation). There is the triangular ruler (at the top left corner of the images) visible in these images, which is further away in the original image, and not captured in the original BEV image, however, it comes into the range of interest as the camera rolls and pitches.

The angles presented in Figure 1.4 is presented in degrees because it is easier to gauge. However, the functions take radians. It is notable that the angles from degrees are converted to angles in Radians. These angles are further multiplied by a factor of 10^{-3}. This factor is taken from the homography matrix. Usually, the Homography matrix is worked in pixels, because it was estimated using checkerboard pattern views in perspective and top view, with the aid of $RANSAC$, therefore it does not explicitly fit to real-world angular measurements. In the third column of the homography matrix, the elements have the highest order of magnitude 10^3. The final element of the matrix is 1. The order of magnitude is related to this element by 10^{-3}.

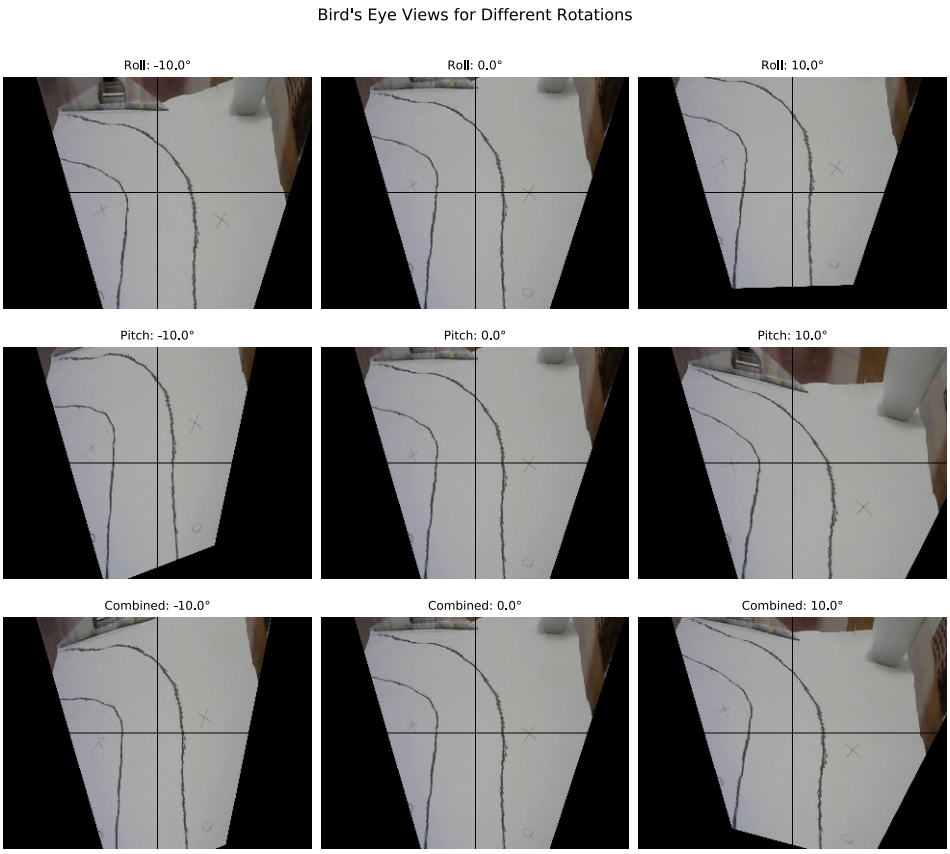

Figure 1.4: BEVs for different camera angles

-

1.4 Lane Detection

As a first step, lane detection is done in a test video, obtained from an internet database. This test video has (a). differently colored line segments (yellow in mid lane separator, and white in the outer lane boundary) meaning a dynamic image texture and (b) the line segment in the middle lane is not always continuous, which requires polynomial fitting to detect lanes as continuous. Figure 1.5 shows a snapshot of the lane detection and polynomial interpolation.

Figure 1.5: Lane detection on test video, perspective scene, and BEV.

1.5 Results from Stereo Camera

Stereo image has disparities in pixels in the image due to the baseline distance between the cameras and thus the visual scene of the camera. The disparity can be used in further image processing. However, we must filter the images strategically for the lane detection part.

An algorithm presented by Turker [2], has given positive and good results in masking outlines detected in dual camera systems. Due to image occlusions and correspondence problems, the disparity image requires some filtering. The process carried out is described by the flow diagram n Figure 1.6 and the results are presented in Figure 1.9.

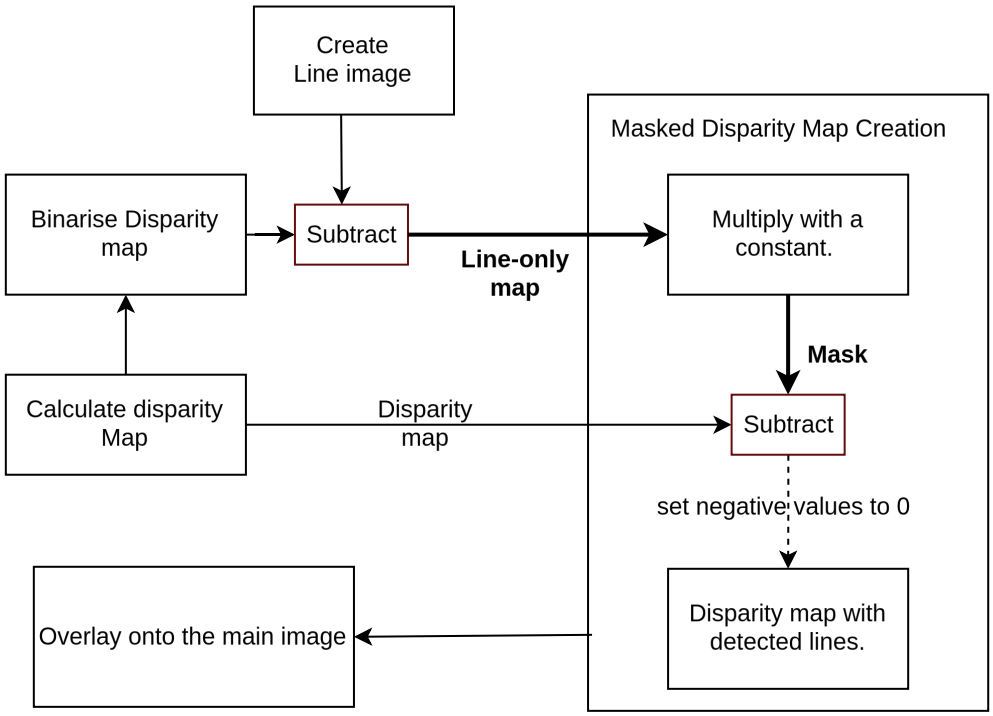

Figure 1.6: A Flowchart of the algorithm.

- **Disparity Map:** The disparity map is calculated from the rectified images from both the cameras left and right. The rectified images are blurred using the Gaussian Blur function to filter out possible noise in the image. Then the openCV function $cv.StereoBM()$ is used to calculate the disparity map. Finally, negative disparity values are removed from the map, by converting them to 0. The map depicts how far away pixels are from each other, from one image to another.

  Correspondence, which is a major problem in stereo vision generally, is visible in disparity maps. Movement of the camera or shift in camera (from one camera to another in our case), means the problem is in finding out points in one image, which

correspond to the same in another image. This is simplified with epipolar geometry, where the problem is reduced to one one-dimensional scanning task, furthermore, filtering, and blurring, makes it a little bit simpler. These techniques are implemented in the Python script as well.

Regarding the disparity and correspondence, we must also find a way to bring the information from two Stereo cameras together into one image, which presents the bird's-eye view of the lane lines and the road. The following steps will try to achieve this goal.

- **Create Line Image:** This step is done to binarise the lines. The image is thresholded from (0, 127) = 0 and (127, 255) = 1. This thresholding gives a 0 and 1 so that the lines have a value of 1. It must be noted that the line image is processed for the image from the right camera.

- **Binarise Disparity map:** In this step, the disparity map is converted to a binary map, with 0s and 1s. This essentially thresholds the disparity map so that very significant disparities are binary coded.

- **Line only map:** The output of subtracting the binary disparity map from the line image is the line-only map. This subtraction isolates the lined areas from regions without lines. However, firstly it must be ensured that the color channels for both the images are the same. This is done in previous steps by converting the image to binary values.

- **Masked Disparity Map:** In this step, firstly the line-only map is multiplied by a constant. Tuerker [2] suggests the value of 100, however, in the case of this report a high value of 300 is used to multiply. This is referred to as a mask, which is subsequently then subtracted from the original disparity map; and finally similar to the disparity map calculation, the negative elements are converted to 0.

- **Masked Disparity Map Overlay:** In this step, the disparity map is overlaid with the original image. The image from the right camera is concerned here, (as the inverted line image is also used from the same). The results of overlaying the masked disparity onto the main image show a nice fit of disparity fitted onto the lanes as seen in Figure 1.9.

With the use of this algorithm, adapted from [2] and modified in this project, the disparity information from the stereo cameras is overlaid onto the detected lanes on the original picture.

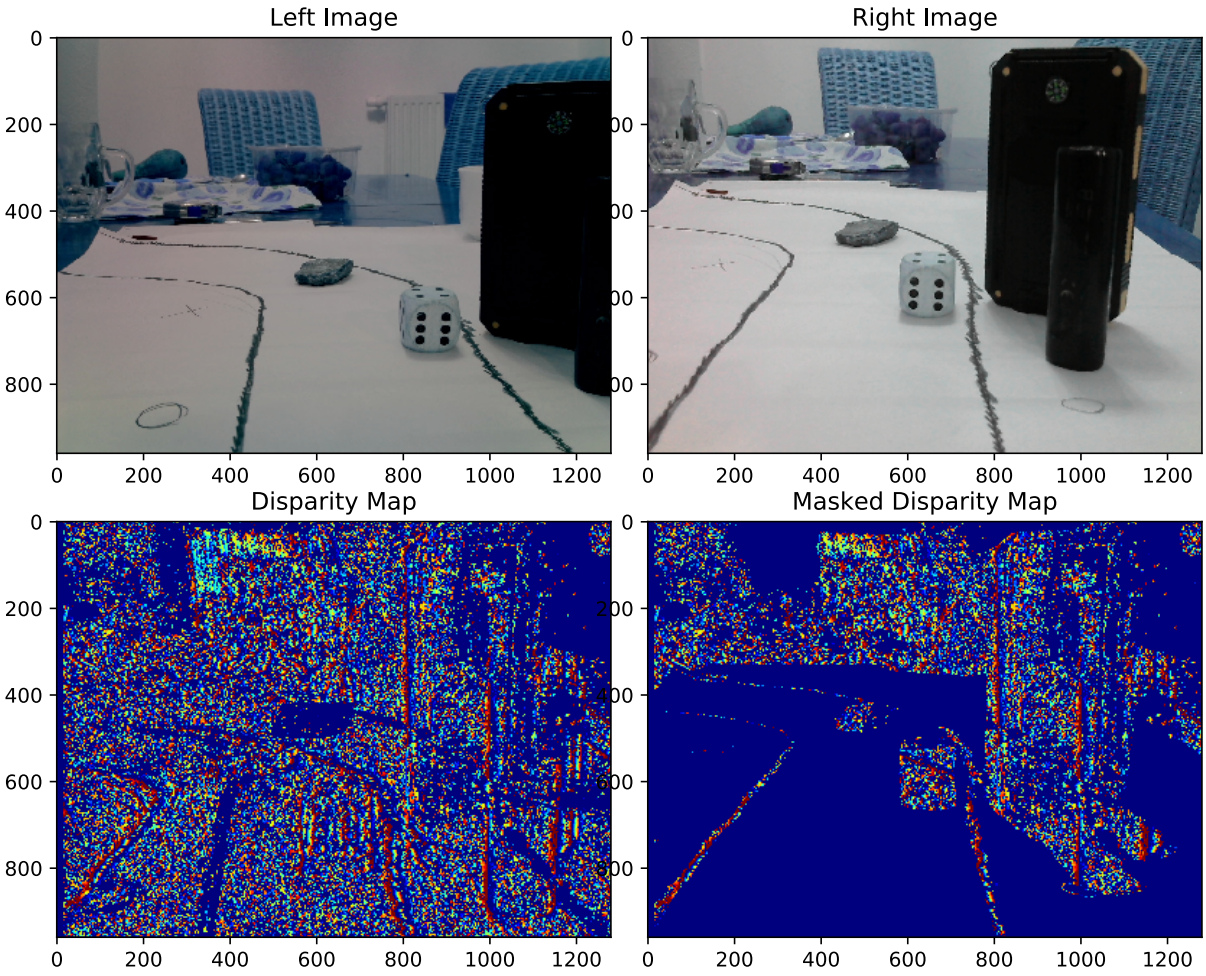

Figure 1.7: Generating Masked Disparity maps on lanes

Figure 1.7 depicts the left and right images, and their disparity map, along with the masked disparity map, overlayed on the right image. After computing the disparity map, areas of high pixel deviation to those of low deviation are distinguishable in color-coded format. The red dots mean high deviation and the lighter colors mean lower deviation (followed according to the enum: $cv.colormapJET$).

For completion, the Bird eye view of the same image is presented in Figure 1.8. In this image, the color map is slightly different. The main image is from the right camera.

The small pebble and a die are visible in the masked disparity image, along with the lane lines, the black blocks, and the objects in the distant horizon. The white surfaces are cleared in the masked disparity image, because they are filtered out with thresholding.

Furthermore, it is important to note, that the mask is created by subtracting the disparity map by a scalar multiple of the line-only map (which is the inverted binary

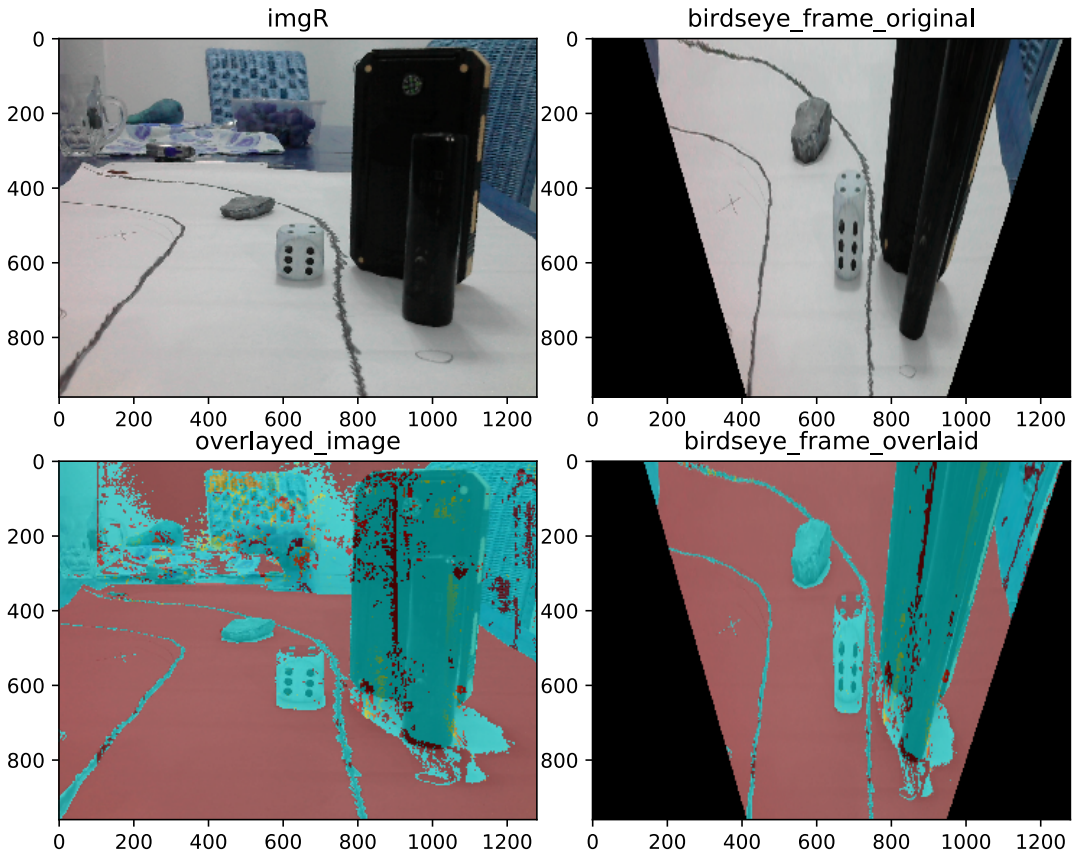

Figure 1.8: Generating Masked Disparity maps on lanes

map of the right image, in this case). This inverted line map, is multiplied here by 300, (compare with [2], who multiples by 100, in MATLAB). The reason for this is that the white surfaces in both the images are most dominant, and we want to filter them out for clarity. So multiplying them by a scalar before subtracting them from the disparity map, ensures that the difference is prominent for the next step.

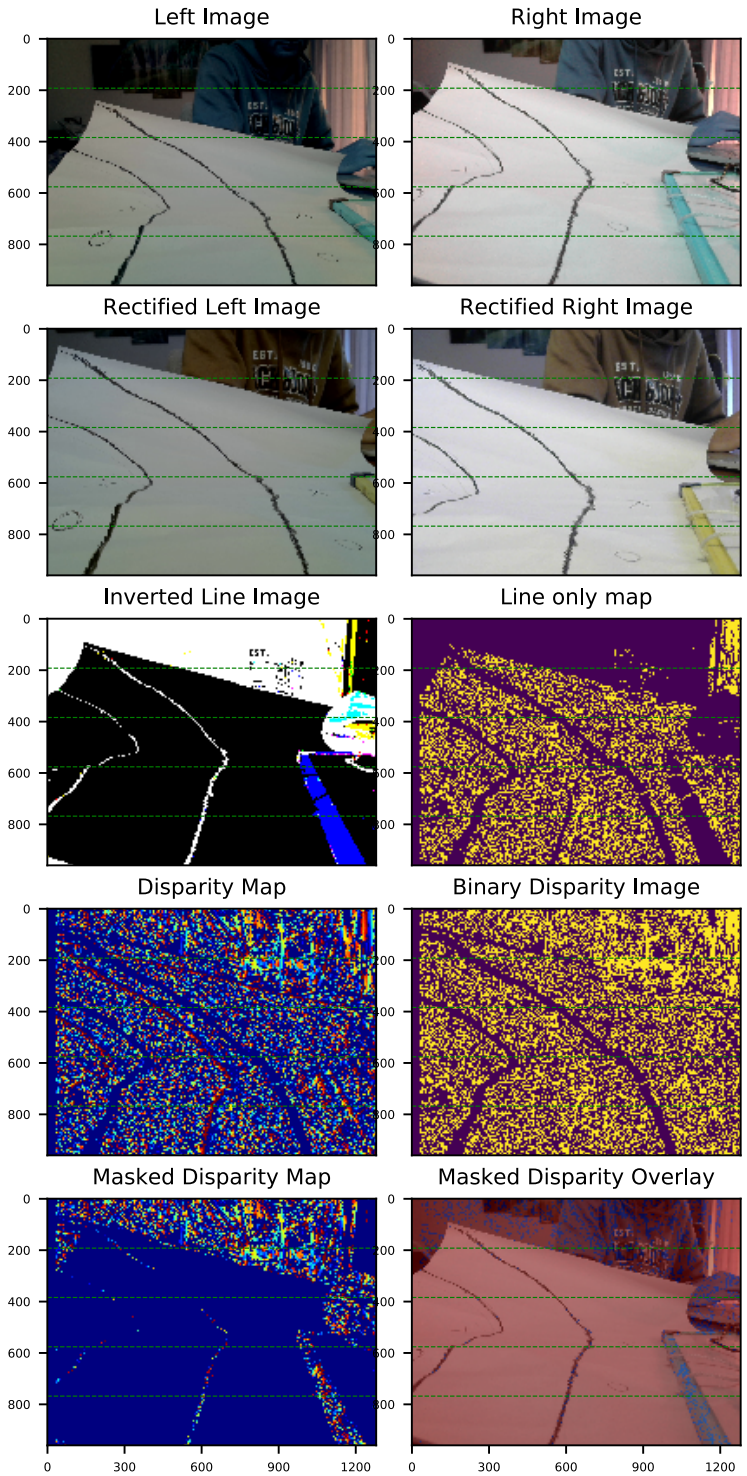

Figure 1.9: Implementation of Turker's algorithm [2]

Bibliography

[1] C. Marvin, “Analyse und implementierung mathematischer methoden für die beschreibung des fahrbahnmarkierungsverlaufs zur realisierung automatischer spurhaltesysteme,” 2022.

[2] T. Alper, “Illumination-robust lane marker detection for stereo inverse perspective mapping,” 2016.