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 Introduction

1.1 Introduction

Today's automobiles are equipped with advanced driver's assistance systems (ADAS), which require comprehensive sensors, cameras, and information fusion and processing in real-time with rather high computation speeds. There are numerous cameras in a modern automobile as depicted in Figure 1.1. Consequently, this requires a significant resource investment by the automobile in processing the environmental data obtained from the images. To achieve image processing for autonomous capabilities, vehicles serve as onboard 'mobile' computers with complex E/E architectures and embedded computers. Integration of cameras, sensors, and data acquisition systems (sensors) and their processing centers (onboard computers in this case) is an interesting domain of investigation.

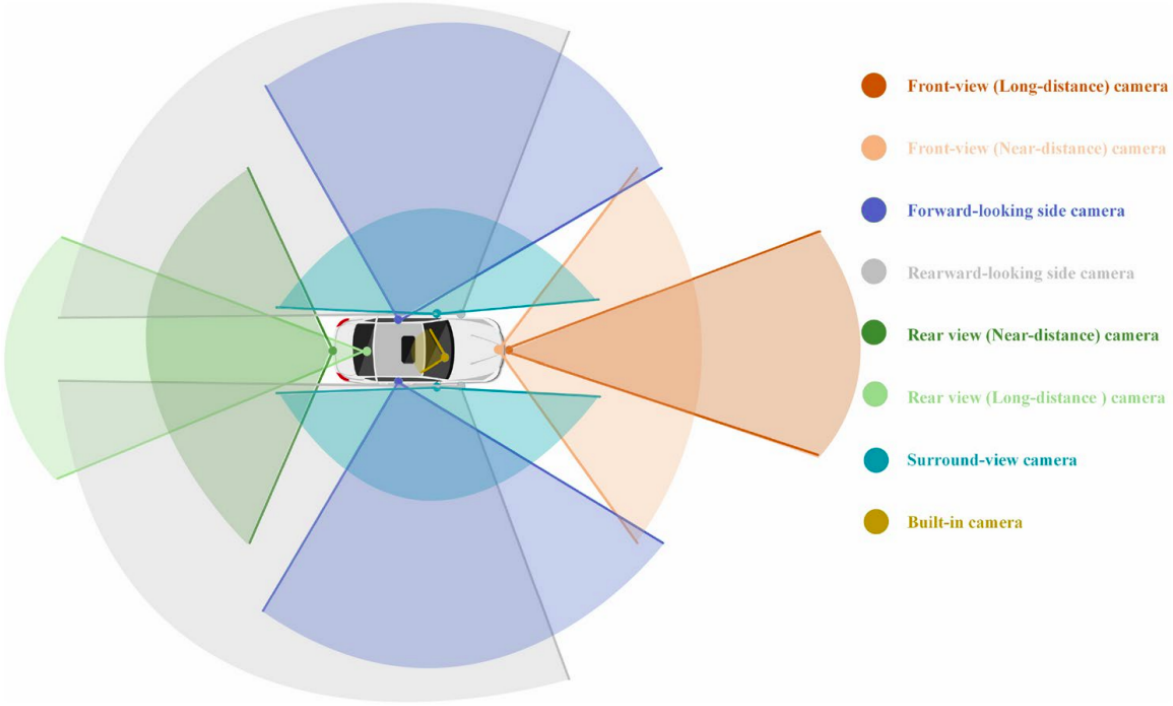


Figure 1.1: Cameras deployed in a modern automobile. [1]

Such modern automotive systems use camera imaging technology as a means to achieve certain goals like automatic cruise control, autonomous driving, and automatic parking assistance to name a few [2]. An important part of this endeavor is the use of camera

systems in processing and transforming the images they take (perspective view) to a Bird's Eye View (BEV), also known as Inverse Perspective Mapping (IPM). Conversion to a bird eye view has some obvious benefits.In a perspective view, lanes converge at a vanishing point in the image, whereas in real life they are parallel. A BEV of images captured helps keep this information intact, paving an effective way for a vehicle's ADAS and other advanced driving features. Furthermore, other features such as potholes can be detected better with a bird eye view in the context of autonomous driving [3]. Figure 1.2 shows an automobile on a roadway with a front camera (blue) and figure 1.3 shows (another car) with a bird eye view.

Figure 1.2: Automobile with front camera capturing images on a roadway [4]

Figure 1.3: An example of an automobile and the environment over Bird Eye View [5]

1.2 Motivation and Objective

The motivation for this work comes from two previous dissertations. Marvin Calligaris has performed his thesis in bird's eye transformation and lane detection in a simulation environment 'Gazebo' using a simulated CAT vehicle [6]. The environment images are captured and processed for lane detection, and bird-eye view transformation. Moreover, Tuerker has implemented inverse perspective mapping for a monocular and a stereo camera in MATLAB and obtained a bird's eye view for images [7]. The two works have in common that the bird's eye view is computed using camera pose (height and angle information) from the perspective view to the bird's eye view.

These works have both computed the bird eye view using geometric relation from the image plane between two camera poses, from the real world. This has a high computational load and can be difficult to implement in real time. Furthermore, vehicle angles are constantly changing during acceleration, deceleration, bumps in roads, and vibrations, effectively undermining the estimated homography that determines the bird's eye view.

From these works, we can provoke some objectives namely

1. Implement a way to obtain Homography to convert perspective view to Bird's eye view, and a way to update this homography in (quasi) real-time, and effectively account for road dynamics in pitch and yaw transformation.
2. Hardware implementation in a mobile platform, such as Raspberry Pi

1.3 Chapter Summary

This chapter introduces the idea of using Inverse Perspective Mapping for images taken from an automobile camera to obtain a bird's eye view for autonomous driving. There are both direct geometric methods for doing this, methods that use feature recognition and image processing in both conventional image processing methods, non-conventional methods that involve learning, and artificial intelligence to process images for the desired end goal.

Moreover, sophisticated tools and systems such as OpenCV and computation platforms like Raspberry Pi allow quick prototyping and research possibilities for computer vision tasks due to the current state of the art.

Investigation in this field is possible in simulation environments like GAZEBO, VREP, or Blender. MATLAB and Python offer tools to perform image processing and bird's eye view transformation, and computation platforms such as Raspberry Pi offer a mobile, standalone computation platform for achieving this task as well. This report therefore aims to do the following:

- Implement image acquisition and transformation from a perspective view to a birds-eye view in a mobile platform with Raspberry Pi as a computer as a real-time system.
- Account for dynamics or changes in camera or vehicle pose by use of an external sensor and fuse the information to update the bird-eye view transform.

2 Background and System Overview

This chapter provides the background information behind the devices, models, formulations and concepts used in the work, including camera models, and various algorithms used for image processing and lane detection. Finally, the devices and tools used in this work are also briefly discussed as a system overview. Details about the software and devices (cameras) used will be provided in Appendix and in the Methodology chapter.

2.1 Camera and Image processing

2.1.1 Monocular Camera Model

A camera is an image acquisition device extensively used as a 'sensor', especially in robotics and autonomous driving. Cameras can be of different types such as pinhole cameras, fisheye cameras, wide-angle cameras, and so on. They have all distinct characteristics and are used as such. A pinhole camera is used in the context of this project. A general mathematical and geometrical model for a pinhole camera is given in the figure below. [8]

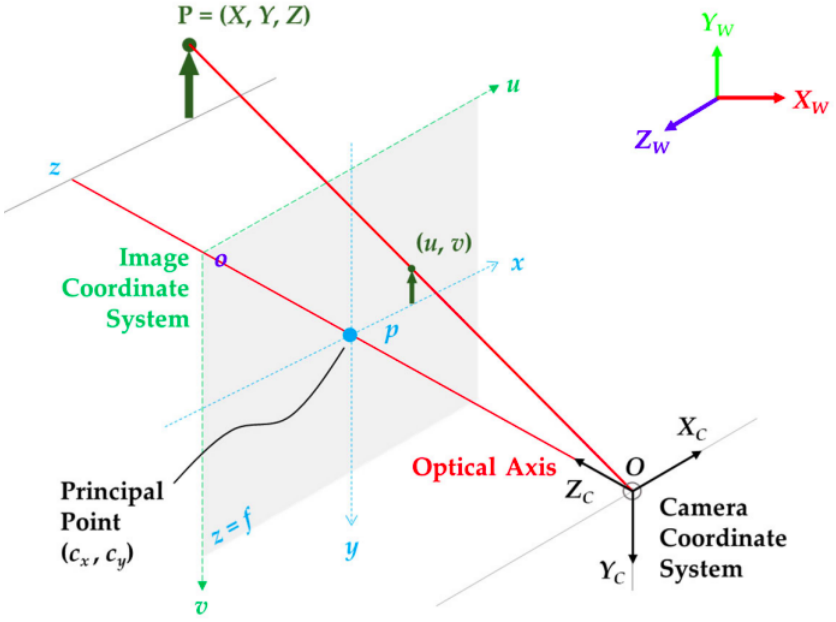


Figure 2.1: Mathematical model of the pinhole camera

In Figure 2.1 above, the world is denoted by the coordinate system $q_w = (x_w, y_w, z_w)$,

the camera coordinate system is denoted by $q_c = (x_c, y_c, z_c)$, and the image coordinate system is denoted by (u, v). The object point represented by $P(X, Y, Z)$ in 3D casts its image onto image plane, (represented by the two green arrows).

The transformation between the world coordinates and the camera coordinates describes the extrinsic parameters of the camera. In other words, the extrinsic parameters of a camera mean how is the camera coordinate system q_c oriented about the coordinate system q_w. This relationship is described by a combination of rotation and translation matrices. In other words, the goal of the modelling is to describe how is the point translated, and rotated from world frame to the camera frame.

The rotation matrix is acquired via rotation of a point about the x y and z axes. This point is also translated by a translation vector t. This encompasses a 3x3 matrix R augmented with the translation vector t to obtain the extrinsic camera matrix.

We have rotation $R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$, and translation $t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$.

This then can be put together to mathematically describe the relationship between the points in the camera frame referred to from the world frame extrinsically.

$$\underline{q_c} = \begin{bmatrix} \underline{R} & \underline{t} \end{bmatrix} \cdot \begin{bmatrix} q_w \end{bmatrix} \tag{2.1}$$

Furthermore, image acquisition in cameras is governed by its intrinsic parameters such as focal lengths, and translation points of the camera intrinsic parameters. These are described by the camera matrix or the camera intrinsic matrix. This is given by C_m. A general form for the camera intrinsic matrix is given by the following equation.

$$C_m = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

Here, f_x and f_y represent the focal lengths in the x and y directions respectively. c_x and c_y represent the coordinates of principal points in the image coordinate frame.

Putting the sequence together, we can write the image in the image coordinate system with

$$q_{image} = C_m \cdot \begin{bmatrix} R & t \end{bmatrix} \cdot \begin{bmatrix} q_w \\ 1 \end{bmatrix} \tag{2.3}$$

It must be noted that the focal length here is for the pixel elements which are given by $f_x = f \cdot s_x$ and $f_y = f \cdot s_y$. The scalers s_y and s_y are the pixel scaler and f is the physical focal length.

From Figure 2.1, depicts an optical axis that goes in the Z_c direction through the image coordinate system (u, v), with the intersection point called the principal point $P(c_x, c_y)$. To account for the difference in image formation away from the principal point in the coordinate system (u, v); we have the following relationship;

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x \cdot \frac{x_c}{z_c} + c_x \\ f_y \cdot \frac{y_c}{z_c} + c_y \end{bmatrix} \tag{2.4}$$

Through the Equations 2.1 to 2.3, we have established the relationship of a camera with coordinates q_c in terms of its pose in the world coordinates q_w , along with its camera matrix C_m and its image projection onto the image coordinate system (u, v). This applies similar to the individual cameras whether it is a monocular, or stereo or even if there are more number of cameras used. However, for more than one camera, the relationship of one camera to another must also be modelled. The following section covers a stereo setup with two individual monocular cameras of the same type.

2.1.2 Stereo Camera model

It is possible, as mentioned earlier to use multiple cameras in a computer vision system. Stereo or dual camera setup is used for some particular reasons. For times when illumination is not robust, a monocular camera might not be as good as a stereo camera setup in autonomous applications. With information from two calibrated sensors, a stereo setup helps mitigate environmental detection inadequacies. Furthermore, stereo camera setups also ensure a provision for depth information mapping wherein scenes and their depth are also captured with the use of two cameras and their disparity information. Disparity is given from the ratio between baseline and focal length.

As the case with monocular setup, the individual cameras follow the same principle, however, with two cameras, it is also of concern for practitioners that the relative rotation, and translation between the two cameras are taken into account, and the images taken by both the cameras be rectified before being used. A stereo setup is modeled mathematically with the aid of figure 2.2.

In the figure we have two cameras, separated by the baseline b, on the left with its origin O_L and on the right with its origin O_R, both capturing the 3D point $P(x, y, z)$, with the image formed on right image plane $P_R(x_R, y_R)$ and the left plane $P_L(x_L, y_L)$. The red line running through the planes, called the Epipolar line is supposed to be parallel or on the same level on both the images. This ensures that the image is rectified and the scenes captured by both cameras are in the same horizontal scan lines, paving the way for further necessary image processing tasks.

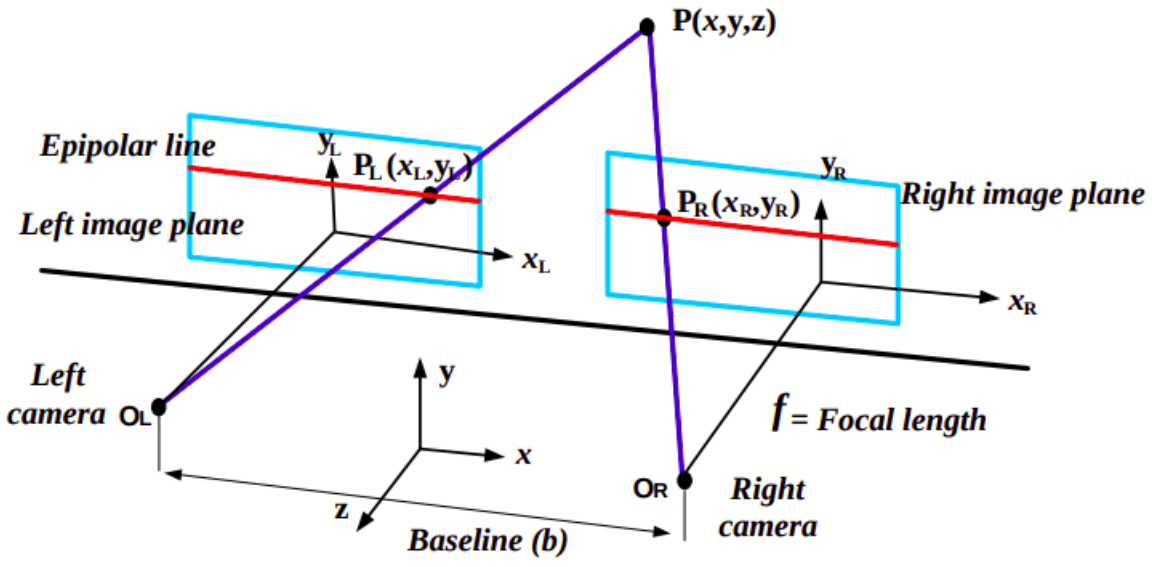


Figure 2.2: Mathematical model of the stereo camera setup [9]

Disparity information from the two cameras helps with gathering depth information. The planar computational space, (planar images or 2D images) need to therefore be projected back to a 3D space for depth to be valid. To achieve this, we need a few projection parameters or matrices. These matrices are also available in openCV. Some useful matrices for the stereo setup and projection of images into depth are presented and explained in Table 2.1.

Parameter	Remark
Rotation and Translation: $\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad ; \mathbf{T} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$	The rotation matrix and translation vectors are defined for the relation between the two cameras. These parameters describe how Camera 2 exists in a 3D space relative to Camera 1. The rotation matrix defines the relative rotation of camera 3 in three principal axes about camera 1; the translation vector represents the linear shift of camera 2 relative to camera 1.
Essential Matrix: $\mathbf{E} = \mathbf{R}\,[\mathbf{T}]_\times \quad \mathbf{T}_\times = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$	The essential matrix relates the two images through rotation and translation, incorporating the skew-symmetric matrix of the translation vector.
Fundamental Matrix $\mathbf{y}^T\mathbf{F}\mathbf{x} = 0$ $\mathbf{x} = \begin{bmatrix} x & y & 1 \end{bmatrix}^T$ $\mathbf{y} = \begin{bmatrix} x' & y' & 1 \end{bmatrix}^T$	$\mathbf{x}$ is the homogeneous coordinate representation of a point in the first image. $\mathbf{y}$ is the homogeneous coordinate representation of the corresponding point in the second image. $\mathbf{F}$ the 3x3 fundamental matrix that encodes the epipolar geometry between the two images.
Projection Matrices P1 and P2: $\mathbf{P1} = \mathbf{C_{m1}} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix}$ $\mathbf{P2} = \mathbf{C_{m2}} \begin{bmatrix} \mathbf{R} & \mathbf{T} \end{bmatrix}$	The projection matrices project 3D object points onto a 2D image plane. Both **P1** and **P2** use the intrinsic camera matrix, but **P2** incorporates the translation relative to camera 1. Here, $\mathbf{C_{m1}}$ and $\mathbf{C_{m2}}$ represent intrinsic matrices of the cameras; $\mathbf{R}$ is the rotation matrix, and $\mathbf{T}$ is the translation vector.
Disparity-to-Depth Mapping Matrix: $\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -\frac{1}{T_x} & \frac{c_x - c'_x}{T_x} \end{bmatrix}$	Transforms disparity values to depth information in 3D space. c_x and c_y represent principal points, f is the focal length, and T_x is the baseline distance between the cameras.

Table 2.1: Generic Forms for OpenCV Matrices

2.1.3 Planar Homography

Image acquisition from a camera can be either capture of an object surface onto an image surface, or capturing an object plane from two or more different camera poses, or capturing the object scene while rotating the camera. In all these cases, the planar surfaces are transformed from one plane to another and are describable using planar homography. Figure 2.3 shows a planar homography relationship between two images and a planar surface.

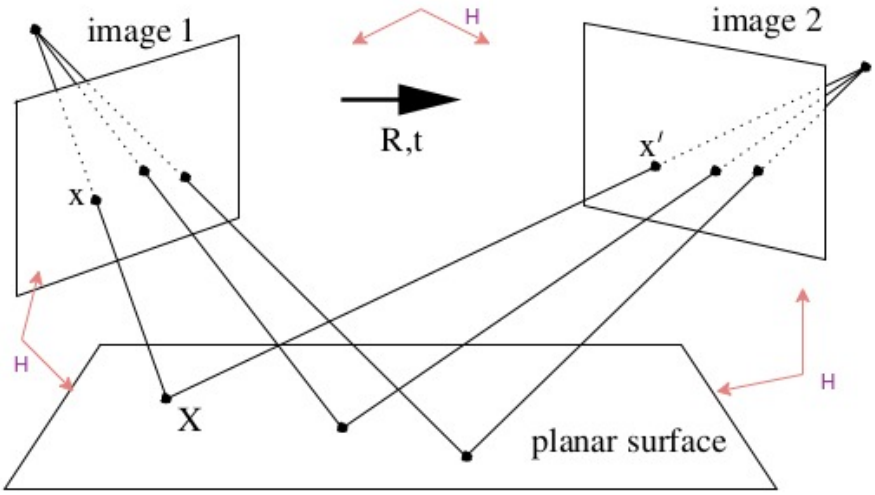


Figure 2.3: Depiction of Homography between image planes [10]

As in Figure 2.1, the points in 3D space (X, Y, Z) can be mapped onto the projection or image plane (x, y) in the projective space $\mathbb{P}^2$. The relationship between the points (x, y) to the points (X, Y, Z) are obtained by a homogenous representation of these points where, $x = \frac{X}{Z}$ and $y = \frac{Y}{Z}$.

A projective transformation is thus introduced by [10] in that in a projective transform, nonsingular $3x3$ matrix H represents a vector x with the given relationship holds.

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Or in simpler form, it can also be written as;

$$x' = Hx \tag{2.5}$$

Previously from Equation 2.3, we have gathered

$$q_{image} = C_m \cdot \begin{bmatrix} R & t \end{bmatrix} \cdot \begin{bmatrix} q_w \\ 1 \end{bmatrix}$$

In this section, the concern is on planar homography, therefore instead of the world coordinate system, it is beneficial to move to the object plane reference. The notation subscript w is also changed to o for this matter in the following equation.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_o \\ Y_o \\ Z_o \\ 1 \end{bmatrix} \tag{2.6}$$

Furthermore, it is also of note that the Z_o component of the object space is 0. This invokes the third row of the rotation matrix to also be 0, essentially leading to the following equation (please note that the scaling is taken to the other side and represented with a new scaler);

$$q_{image} = s' \cdot C_m \cdot \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \cdot \begin{pmatrix} X_o \\ Y_o \\ 1 \end{pmatrix} \tag{2.7}$$

The elements $s' \cdot C_m \cdot [r_1 r_2 t]$ can be encapsulated into a 3x3 matrix H and be re-written in the form similar to Equation 2.5. This completes the formulation of planer homography for images from the object plane to the image plane.

$$q_{image} = H \cdot \begin{pmatrix} X_o \\ Y_o \\ 1 \end{pmatrix} \tag{2.8}$$

Although the Homography formulation is presented here with the help of camera intrinsic and extrinsic, (model based approach), other techniques exist where features of known entities such as checkerboard pattern are captured to obtain the homography between images. The practical aspect is discussed in chapter **??**.

2.2 Lane Detection

Lane detection is done via image processing techniques where lane lines or limits are used to detect boundaries of the road. The lane lines in real life are normally white-colored lines on black asphalt, effectively providing contrast and helping the detection algorithms. As the image is taken, it is usually blurred for good assessment. This is done using Gaussian blur function.

Gaussian Blur:
A Gaussian blur is applied to the image, to smoothen out the noise and light, unwanted edges or 'features'. Gaussian blur can be described mathematically as follows,

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

In image processing applications, in 2D arranged pixels, the Gaussian blur uses a square kernel, with the main pixel at its center, which uses the σ value for blur intensity (the range of bell curve in the Gaussian curve), on all sides, and provides an output image, running through the pixels in the image, in turn providing a smoothened out image in the end that reduces unwanted noise while keeping the needful and distinctive features intact. Gaussian blur also helps ensure that the focus is on the intensity in the images, which reduces computation time. [11]

Edge Detection (Canny):
An image has a lot of non-useful properties and features that are better filtered out for processing purposes. This reduces the workload on the processor as well as makes computation less complex, organized, and faster. An application is the detection of edges, which is essential in lane detection applications. One of the most widely used edge detection algorithms is the Canny edge detector [12].

Canny edge detector [13] firstly takes the input image and calculates the gradients around each pixel in both the x and y directions. The algorithm then finds the direction of edges, by using the gradient information. It then looks at the individual pixels and their surrounding pixels in the gradient direction and keeps the pixels that are the brightest in the direction of the gradient, in a process called non-maximum suppression. In other words, non-maximum pixels are suppressed (mostly in the gradient direction).
Furthermore, it must also be considered that Canny edge detection uses hysteresis values or the idea of thresholding. A high and a lower threshold are to be provided, such that pixels below the lower threshold values are rejected as non-edges, pixels above

the threshold are kept as certain edges, and pixels lying in between the thresholds are taken as either edges if they have neighbors that are edges or are rejected if it is not the case.

Hough Line Transform:
Although edge detection gives an indication that these are lanes in a lane detection application, for instance, it is not certain. Another step is to be performed to ensure that detected edges are indeed lanes in the context of autonomous driving. A useful algorithm for line transformation is the Hough line transform [14], used to detect lines in images.

The basic idea is to transform an image space (in a cartesian coordinate system) into a parameter space or Hough space (in polar coordinates) wherein points (in the parameter space) that represent lines (in the image space) are mapped to sinusoidal curves. As these curves intersect in the parameter space, the presence of lines in the image is indicated.

For instance, from edge detection, we get the pixels that represent edges on an image. These are taken as potential lane lines. These lines are fed through the Hough line transform algorithm, with the end goal that the output is a line that passes through the edge pixels. The lines with the most intersections in the parameter space are indicative of lane lines.

A line in Cartesian system, $y = mx + c$ can be represented in polar coordinate system as $r_\theta = x_0 \cdot \cos\theta + y_0 \cdot \sin\theta$. For the points (x, y) in Cartesian coordinates, when transformed to curves in the parameter space or polar coordinates, the curves if they intersect, mean that they are part of the same line. Figure 2.4 depicts this idea.

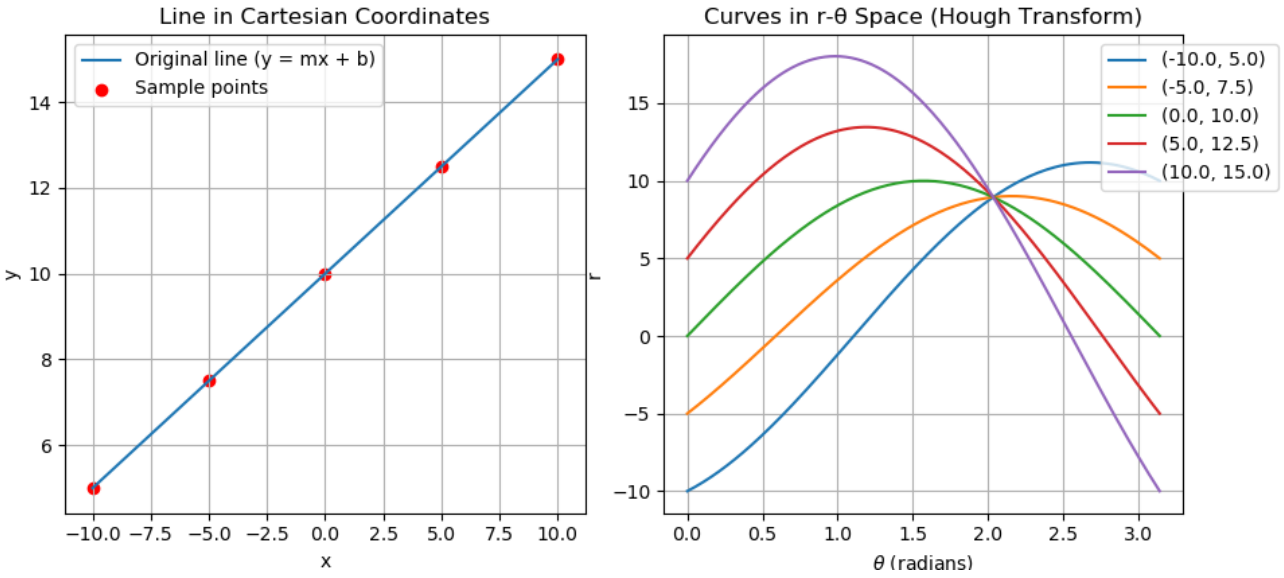


Figure 2.4: Depiction of Hough line transformation

Furthermore, the Hough line transform gives two points for making a line, $[(x_1, y_1), (x_2, y_2)]$.

This gives a slope m as well. With a series of these points, polynomial fit can be done, which essentially draws a line that makes the best fit, linear, quadratic or of higher order.

Sliding Window Algorithm

Continuous video feed needs to be segmented into smaller chunks to be processed in real time, especially in applications like lane detection. A way to achieve this is by using sliding window algorithm, which essentially, as the name suggests, slides through a series of data (such as arrays, lists or text for instance), selects a section at a time, and implements processing inside it, and moves on to the next segment of the window.

Sliding window method is a handy computational tool as it can work in spatial as well as temporal continuum with its systematic manner of moving through series of information. The complexity of this algorithm is $O(n)$; as it adds and removes one element at a time in each subsequent block. [15]

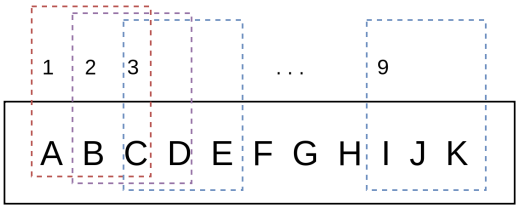


Figure 2.5: Sliding Window Algorithm Example

Figure 2.5 shows a simple sliding window algorithm that runs through 11 alphabets, and 'processes' them. The window takes in 3 elements at a time, does the processing part in this block, and then moves to the next window block in the subsequent step. This subsequent block contains the next element in the data series, and to keep the block size of 3, it removes the last (leftmost entry).

2.3 Chapter Summary

This chapter provides some relevant background information on the ideas and concepts used behind this work.

The mathematical model for monocular pinhole camera is described. The relationship between the intrinsic parameters of a camera, to the 3D plane, and its mapping back to the image plane, (image formation) is discussed, with the help of equations. Furthermore the idea of Planar Homography was also described where the idea that two images of the same point can be related by a homography is introduced. This paves way for Homographic transformation which helps in obtaining a Bird eye view, as will be seen in the following chapters. In this context, a 3x3 Homography matrix is also introduced.

A model of the stereo camera model is also presented in this chapter. The individual cameras follow the same principles as the monocular setup, however, in the stereo setup, the relative information about the cameras is of essence. Since two images will be on different planes when they're taken from different camera pose,the idea of epipolar lines is introduced, and with it, the possibility of rectification of the image taken from two cameras, for further processing. Important matrices are introduced and briefly remarked in Table 2.1.

Finally, the theoretical ideas behind the methods such as edge detection (Canny) and Hough line transform are described. A sliding window algorithm is introduced, as it has been a nice computational technique to select best points and use for processing. These techniques will be used later as openCV functions in the upcoming chapters.

In the following chapter, it will be shown how these concepts come into play during application.

3 Methodology

3.1 Camera Calibration

This section describes the process of finding the focal length, principal points and distortion parameters of the camera used, also called the camera calibration process, using a checkerboard pattern.

3.1.1 Monocular camera calibration

Calibration of the single or monocular camera is done as depicted in Figure 3.1.

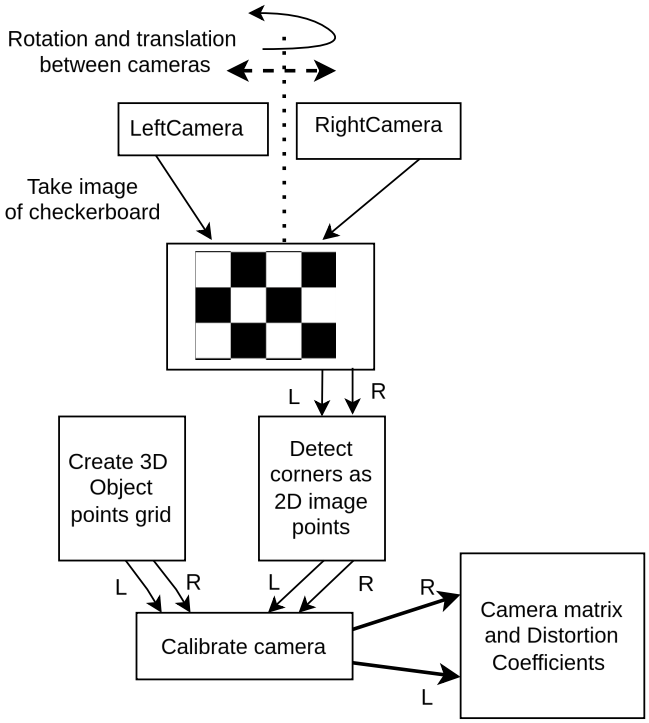


Figure 3.1: Monocular camera calibration

- **Image Acquisition:** The images were acquired using the OpenCV function using a Logitech C270 Webcam. The calibration step was performed using 15 images of a chessboard pattern of size 7x6 [internal corner size of (6,5)]. Initially, a square pattern of size 9x9 was used, however, due to the square nature and small square sizes, the results were not accurate. To avoid redundant rotation, a rectangular chessboard size was used.

- **Processing for camera intrinsics**: OpenCV functions were used to estimate the intrinsics of the camera. The process also involved minimizing the re-projection error (The difference between observed image points and projected 3D points in the image). The python script used for the camera calibration part uses the openCV libraries extensively.

- **Results**: The camera calibration process effectively gave the camera intrinsic matrix, known as the camera matrix, the distortion coefficients (lens distortion of the cameras). These calibrated parameters were used later in subsequent steps. The results are presented in the following chapter.

3.1.2 Stereo camera Calibration

The stereo camera calibration process is depicted by Figure 3.2.

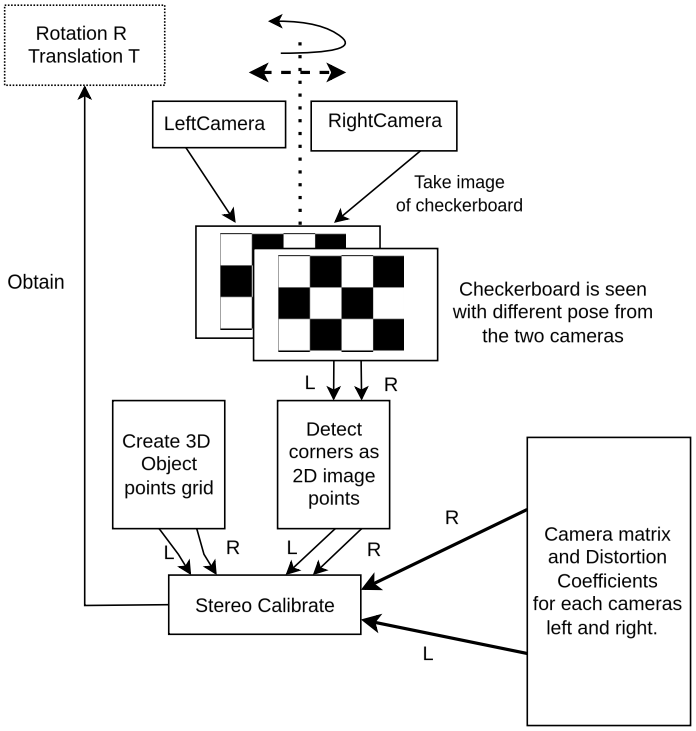


Figure 3.2: Stereo setup calibration

To calibrate a stereo setup, the checkerboard pattern and its detected corners are used again, like in the monocular camera setup. However, these points are used to determine the relative rotation and translation (**R** and **T**) between the cameras. Moreover, the calibration process also needs the individual camera matrices as parameters, which are fed from the previous step. As discussed in chapter 2, the stereo camera setup is concerned with the rotation and translation of a camera related to its stereo counterpart. The output of this calibration process is the rotation and translation matrices (**R** and **T**) of the cameras relative to each other. To achieve this, Figure 3.2 depicts the general algorithm

of doing the stereo calibration.

When we talk about the parameters of rotation (**R**) and translation (**T**), we are talking about the extrinsic parameters. The world coordinate system is in this case used as the coordinate system of camera 1. And the two matrices (**R** and **T**), are derived from this datum. However, it is okay to use either of the two cameras. In this case, the right camera is used as the main camera throughout the project and is assumed to be the origin of the world coordinate system.

After the cameras are calibrated, they need to be rectified, i.e. the epipolar lines need to be colinear, so that the pixels corresponding to the same scene lie in the same scanning line for further image processing such as disparity calculation. The steps in rectification of the image is depicted in Figure 3.3.

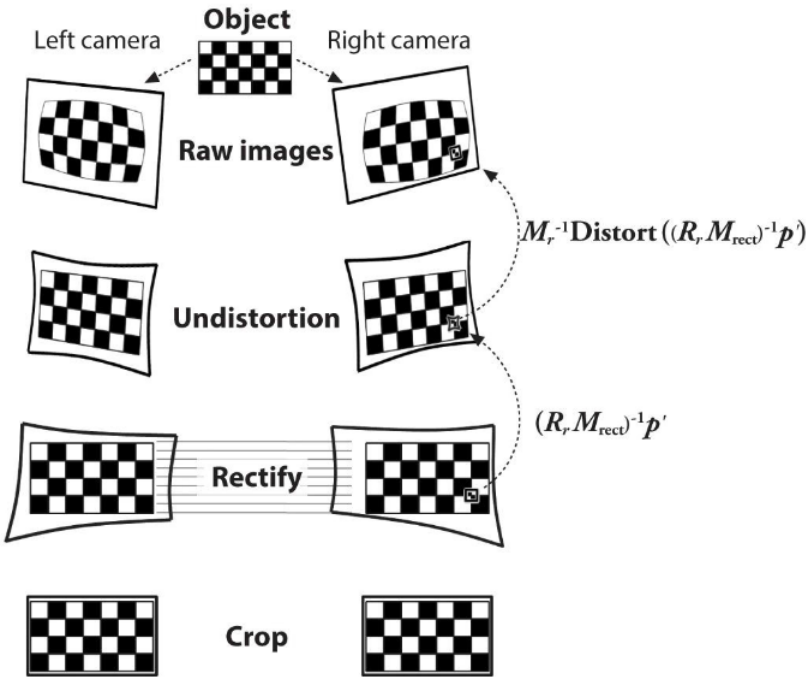


Figure 3.3: Stereo Camera Calibration depiction [16]

After the cameras are calibrated, they are ready for further processing.

3.2 Bird Eye View transform and Lane Detection

3.2.1 Bird's Eye View Transform

Different methods exist to obtain the Bird eye view of an image taken in perspective view. Firstly the method using homography matrix estimation is presented and second, an adaptation from Calligaris [6] is presented. Furthermore, for changes in camera pose (angles), a proposition of using an IMU sensor to gather the angular change in the system (R_c) to transform the homography matrix further accordingly, is presented. A general pipeline is depicted in Figure 3.4. The IMU sensor used in this project is not a stand-alone sensor, but rather a wafer with other sensors, embedded in an Arduino controller (Arduino Nicla Sense ME). The sensor gives out the desired values for the three angles.

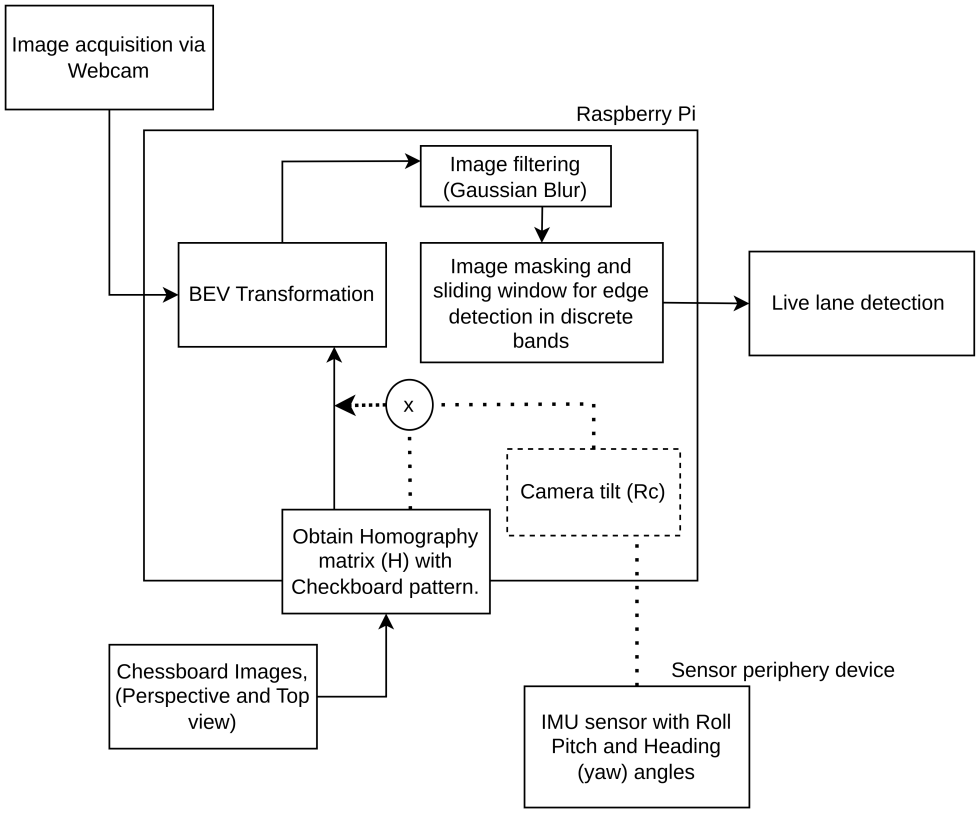
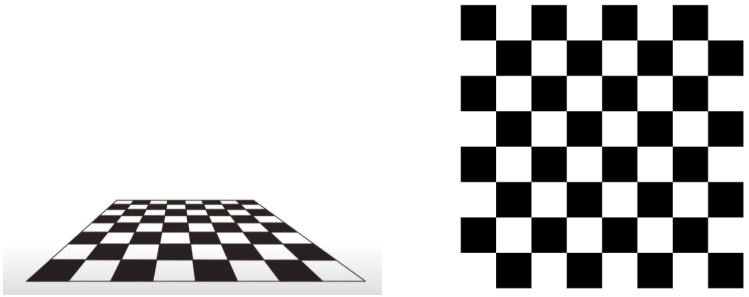


Figure 3.4: Depiction of the general methodology in the project

In this work, two images of a checkerboard pattern of size (7,6) [internal corner size of (6,5)] are captured in perspective view and in a top-down view. The effective size of the pattern is (6,5) because the process is concerned with the inner corners given by $(n-1)$ for n corners in each row or coloumn. Figure 3.5 shows a generic set up of the two views of the checkerboard pattern.

The *cv.cornerSubPix* algorithm is used to detect the checkerboard corners in both



(a) Perspective View (b) Top view

Figure 3.5: Checkerboard pattern views for estimating homography

images. With these detected corners, the change of pose in the camera from the perspective to the bird eye view or the top view is calculated by the function $cv.findHomography()$ using the $RANSAC$ algorithm. The output of this function is the 3x3 Homography matrix given by,

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

3.2.2 Lane detection and tracking

Inverse perspective transform results in the lane lines on a roadway to be parallel in the image plane. This gives a nice top-down view to do lane detection and tracking on the road. Figure 3.6 shows the process.

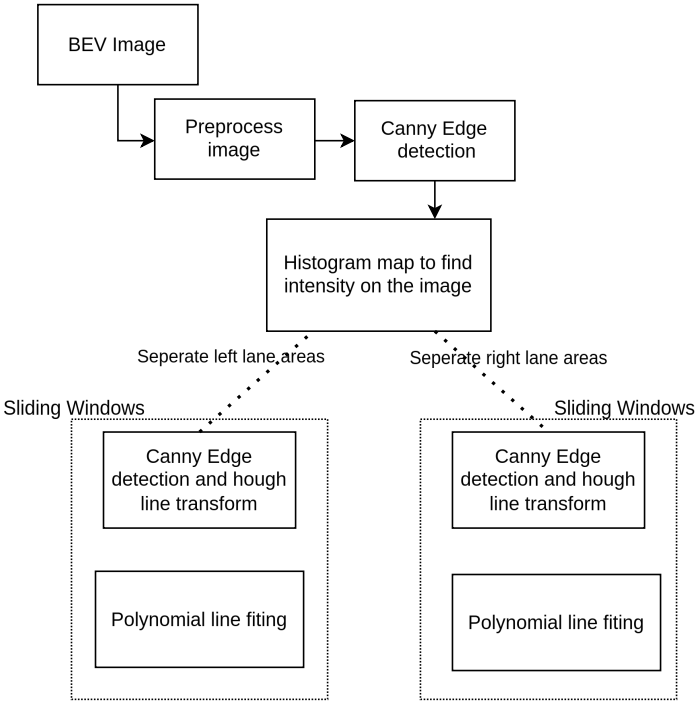


Figure 3.6: Lane detection pipeline

A video input from a test video or a webcam is read. In this, a histogram is evaluated, effectively giving us the location of pixels corresponding to lanes. Then along the perpendicular of these pixels, lanes are present.

Each frame of the input video is separated into sliding windows. Inside of each of these sliding windows, a Canny edge detection is done, and the detected lines are used to perform a Hough line transform to ensure lane lines. These are then plotted back onto the bird's eye view image. Furthermore, using the values obtained from the Hough line transform, a polynomial fit of these points ane is done, to predict the lane for the overall window.

Bibliography

[1] C. Wang, X. Wang, and H. e. a. Hu, "On the application of cameras used in autonomous vehicles," *Archives of Computational Methods in Engineering*, vol. 29, no. 8, pp. 4319–4339, 2022. [Online]. Available: https://doi.org/10.1007/s11831-022-09741-8

[2] C. Kühling, "Fisheye camera system calibration for automotive applications," Master's thesis, Freie Universität Berlin, 2017.

[3] T. Zhao, L. Yang, Y. Xie, M. Ding, M. Tomizuka, and Y. Wei, "Roadbev: Road surface reconstruction in bird's eye view," 2024. [Online]. Available: https://arxiv.org/abs/2404.06605

[4] "Lane departure warning system market | 2018-2024 (denso, bosch, delphi, autoliv, mobileye, magna international, continental, zf group, wabco)," December 2018. [Online]. Available: https://www.openpr.com/news/1439285/lane-departure-warning-system-market-2018-2024-denso-bosch-delphi-autoliv-mobileye-magna-inter html

[5] I. S. AMS, "3 types of 3d sensing for smartphones and self-driving cars," *IEEE Spectrum*, May 2019. [Online]. Available: https://spectrum.ieee.org/sponsored/ams/3-types-of-3d-sensing-for-smartphones-and-self-driving-cars

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

[7] T. Alper, "Illumination-robust lane marker detection for stereo inverse perspective mapping," 2016.

[8] D. J. Yeong, G. Velasco-Hernandez, J. Barry, and J. Walsh, "Sensor and sensor fusion technology in autonomous vehicles: A review," p. 2140, 03 2021.

[9] M. Al-Azawi, D. Al-Rawy, S. Al-Jobory, and D. Zaghar, "Stereo vision for 3d measurement in robot systems," *Journal of Engineering and Sustainable Development*, vol. 17, 03 2019.

[10] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. USA: Cambridge University Press, 2003.

[11] "Canny edge detection explained: A step-by-step guide," accessed February 22, 2024. [Online]. Available: https://medium.com/@nadeeshamadusanka44/canny-edge-detection-explained-a-step-by-step-guide-f51dad94fa9b

[12] J. C. Niebles and R. Krishna, "Lecture: Edge detection," accessed February 27, 2024. [Online]. Available: http://vision.stanford.edu/teaching/cs131_fall1718/files/05_edges.pdf

[13] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.

[14] P. C. Hough, "Method and means for recognizing complex patterns," *IRE transactions on electronic computers*, vol. EC-11, no. 1, pp. 80–86, 1962.

[15] E. Gul, "Sliding window algorithm explained," 2024. [Online]. Available: https://builtin.com/data-science/sliding-window-algorithm

[16] S. E. C. Gopiraj, "Calibration and rectification of a stereo camera with opencv." [Online]. Available: https://stackoverflow.com/questions/53394508/