Segmentation module for autonomous car

Student: Rahil Amit Shah
Roll Number: EE20B104

Progress update on the project

Guide: Dr. Ramkrishna Pasumarthy

Dept: Electrical Engineering
IIT Madras
India
Date: April 1, 2024

Contents

Current progress

1. Implemented the YOLOP model for segmentation, object detection, and lane detection.
2. Further extracted the pixel values of drivable coordinates from the image.
3. Estimated the depth using the ZED's point cloud data from the extracted pixel values.

The flow of the experiment is summarized in the flow chart below.

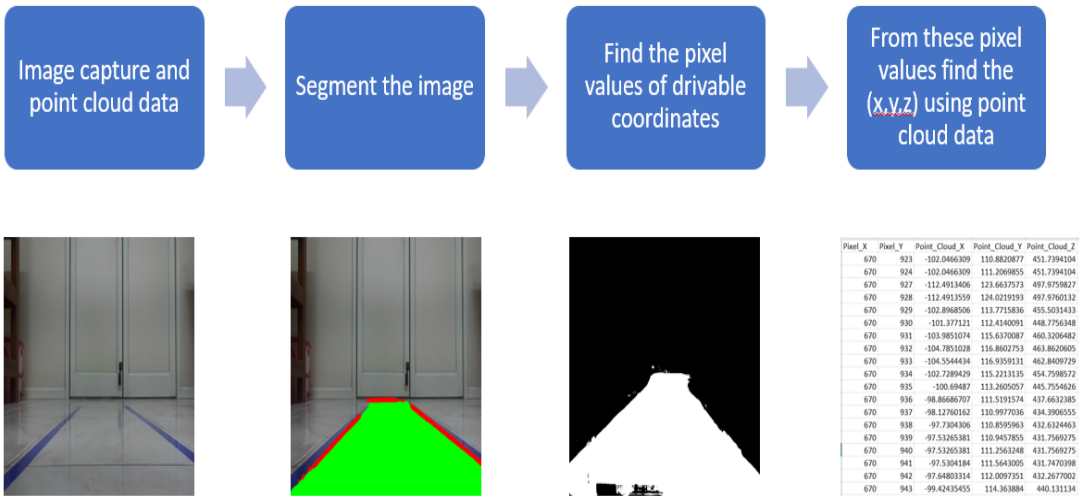

Figure 1: Flow chart

Upon thorough examination, it became evident that the estimation of depth provided by the ZED system was significantly erroneous compared to the ground truth values. Subsequently, in the subsequent section, our efforts were directed towards delineating the specific areas and reasons behind these inaccuracies, alongside identifying instances where the ZED system functioned reliably.

Consequently, our attention pivoted from the module itself to the crucial task of attaining accurate depth estimates from the images. It is imperative to delineate the significance of depth estimation in this context.

Depth estimation plays a pivotal role in various computer vision applications, particularly in autonomous driving systems. Accurate depth information is fundamental for tasks such as obstacle detection, path planning, and environment mapping. Without precise depth estimation, the system's ability to perceive and navigate through its surroundings is severely compromised, posing significant safety risks and hindering the efficacy of the entire system.

Hence, achieving accurate depth estimation is not merely a technical endeavor but a fundamental necessity for ensuring the reliability and safety of autonomous systems operating in real-world environments.

The Depth Estimation Problem with Zed

We conducted a series of experiments to quantify the depth of the image using the ZED's point cloud approach.

The experiment procedure is outlined as follows:

1. A scale is positioned on the ground to serve as a reference.
2. The camera is placed at a height of 5.5 cm from the ground, mirroring its current placement on the vehicle.
3. Depth values are estimated within our region of interest, specifically the area between the tracks, with a width of 40 cm.

The experimental setup is visually depicted in the image below.

Figure 2: Experimental setup

Upon analysis, it was observed that the depth estimation of points on the ground yielded highly inaccurate results, often resulting in NaN (Not a Number) values. A screenshot of the collected data illustrating this issue is presented below:

Figure 3: Reference of pixel values for an image

Pixel_x	Pixel_y	X	Y	Z
684	936	nan	nan	nan
645	920	nan	nan	nan
684	889	nan	nan	nan
711	862	nan	nan	nan
852	882	-61.7	140.4	650.2
852	882	-62	140.9	652.7
819	844	-78.9	125.4	664.9
792	848	-93	128.9	673.5
1257	904	nan	nan	nan
1248	889	124.6	145	656.4
1401	907	nan	nan	nan
1401	907	nan	nan	nan
1401	907	nan	nan	nan
1401	907	nan	nan	nan
1401	907	nan	nan	nan
1401	907	nan	nan	nan
1401	907	nan	nan	nan
1320	862	149.8	125	620.1

Figure 4: Data collected from zed

This discrepancy in depth estimation undermines the reliability of the ZED system, particularly in scenarios where accurate ground-level depth information is crucial for navigation and obstacle detection. Consequently, it becomes imperative to explore potential solutions to rectify the issues associated with the ZED system.

In the subsequent sections, we will delve into various strategies and techniques aimed at addressing the shortcomings of the ZED system and enhancing its accuracy and reliability in providing ground-level depth information. By identifying and implementing effective solutions, we aim to mitigate

the impact of these discrepancies and bolster the performance of the ZED system in real-world applications.

Solution to the depth problem

Within this discussion, we explore two potential solutions to address the depth estimation challenges. The first solution revolves around leveraging the mathematical model underlying the physics of depth estimation, commonly known as inverse perspective mapping. This method aims to derive depth values for ground-level objects captured by the camera. By employing inverse perspective mapping, we intend to refine the accuracy of depth estimation, particularly for objects situated close to the ground.

The second solution focuses on obtaining depth information for objects positioned at a distance from the camera. This aspect of depth estimation is efficiently handled by the ZED system, which excels in capturing depth data for distant objects. By leveraging the capabilities of the ZED system in this regard, we aim to enhance the overall depth estimation process, ensuring accurate and reliable depth measurements for objects located at varying distances from the camera.

Inverse perspective mapping

Pinhole camera model

Imagine placing the image sensor inside a confined space, encapsulated within a box featuring an incredibly small pinhole, also known as an aperture. This setup constitutes the essence of a pinhole camera.

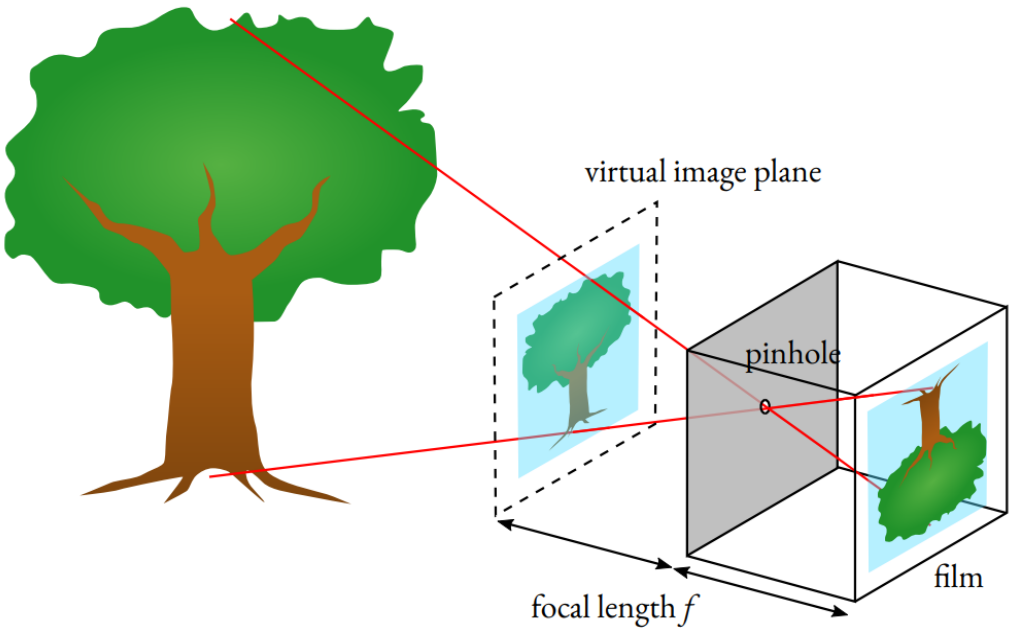

Figure 5: Pinhole camera

In the realm of image formation, the concept of an ideal pinhole serves as a fundamental approximation. In this idealized scenario, the majority of incoming light rays are effectively blocked, allowing only a single ray to reach each point on the image sensor. Despite resulting in an upside-down image, this inversion does not pose significant concerns for our purposes.

However, in reality, the pinhole cannot be infinitesimally small due to practical constraints. If the aperture were too tiny, insufficient light would enter the camera, leading to dimly lit images. Conversely, enlarging the aperture beyond a certain threshold introduces issues of blur, as light rays from different angles converge onto the same sensor point, causing a loss of image sharpness. To counteract this, real-world cameras incorporate lenses to focus incoming light onto the sensor, mitigating blur.

Although our discussion centers on the pinhole camera model, which does not account for lens effects, it serves as a remarkably accurate approximation for cameras with lenses. Therefore, in subsequent discussions, we maintain the simplification of an ideal pinhole, recognizing its practicality and effectiveness in modeling real-world camera systems.

Visualizing this concept, the image plane is introduced as a fundamental component. It serves as the surface onto which light rays converge, forming the captured image. Through this model, we gain insights into the principles governing image formation, paving the way for deeper understanding and analysis of camera systems.

In the context of the pinhole camera model, the image sensor resides at a distance f behind the pinhole, constituting the so-called "image plane." Contrarily, an imaginary construction known as the "image plane" is positioned f in front of the pinhole. This configuration enables us to relate the size h' of an object to its size h on the image sensor through the principle of similar triangles.

Mathematically, this relationship is expressed by the equation:

$$\frac{h'}{h} = \frac{f}{d}$$

This equation, denoted as Eq. eq-hprime-over-h, elucidates that the size h' of the object in the image diminishes as the distance d to the camera increases. Consequently, distant objects appear smaller in the captured image, a phenomenon commonly observed in photography and image processing.

To generalize Eq. eq-hprime-over-h, we introduce coordinates x and y within the image plane, as depicted in the accompanying sketch.

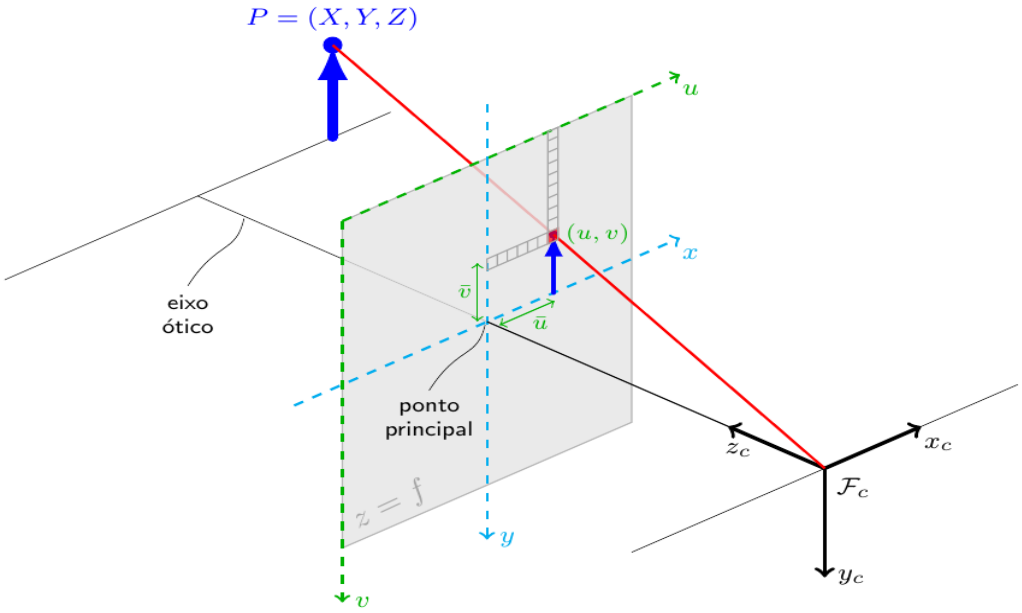

Figure 6: Image formation

The origin of the camera coordinate system (X_c, Y_c, Z_c) is at the location of the pinhole. The gray shaded region is the part of the image plane that gets captured on the image sensor. The coordinates (u, v) are the pixel coordinates that were already introduced.

The sketch allows us to determine the mapping of a three-dimensional point $P = (X_c, Y_c, Z_c)$ in the camera reference frame to a two-dimensional point $p = (x, y)$ in the image plane. We have a look at the above figure in the Z_c-X_c plane:

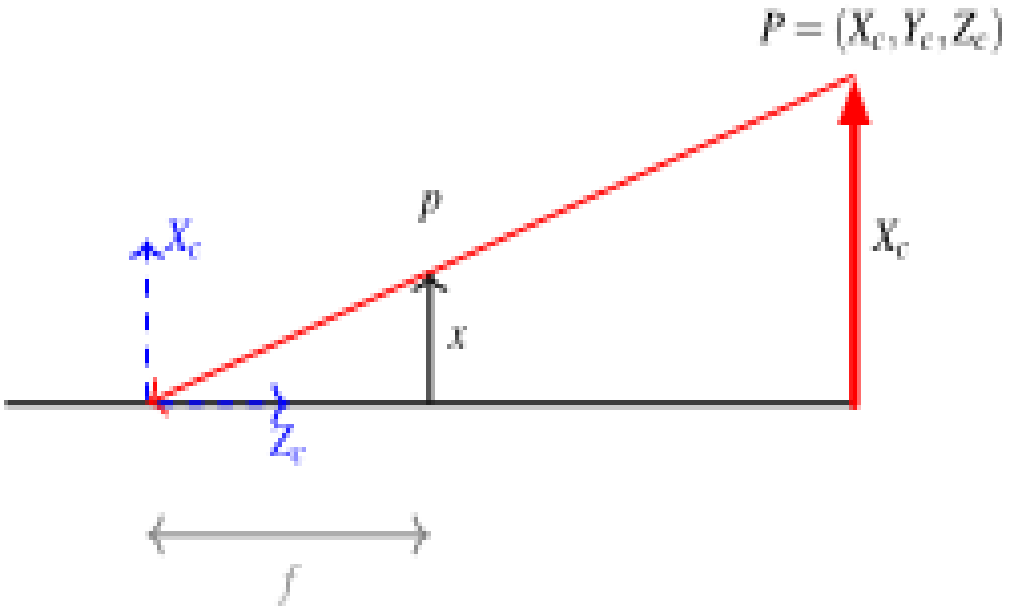

Figure 7: Pinhole camera

Similar triangles in the above figure reveal

$$\frac{x}{f} = \frac{X_c}{Z_c} \Rightarrow x = f\frac{X_c}{Z_c}$$

And similarly we find

$$y = f\frac{Y_c}{Z_c}$$

Now, we want to establish what pixel coordinates (u, v) correspond to these values of (x, y) (cf. **??**). First, we note that the so-called *principal point* $(x = 0, y = 0)$ has the pixel coordinates $(u_0, v_0) = (W/2, H/2)$. Since x and y are measured in meters, we need to know the width and height of one "sensor pixel" in the image plane measured in meters. If the pixel width in meters is k_u and the pixel height is k_v, then

$$\begin{aligned} u &= u_0 + \frac{1}{k_u}x & &= u_0 + \frac{f}{k_u}\frac{X_c}{Z_c} \\ v &= v_0 + \frac{1}{k_v}y & &= v_0 + \frac{f}{k_v}\frac{Y_c}{Z_c} \end{aligned}$$

If we define $\alpha_u = f/k_u$ and $\alpha_v = f/k_v$, we can formulate the above equations in matrix form

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \frac{1}{Z_c}\begin{pmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix}$$

Typically $\alpha_u = \alpha_v = \alpha$.

To continue our discussion, we need to introduce the concept of homogeneous coordinates.

Reference frames

In our applications, we will deal with different coordinate systems, so we will not always have our point P given in coordinates of the camera reference frame $(X_c, Y_c, Z_c)^T$. For our lane-detection system, the following reference frames are relevant:

- World frame $(X_w, Y_w, Z_w)^T$
- Camera frame $(X_c, Y_c, Z_c)^T$
- Default camera frame $(X_d, Y_d, Z_d)^T$
- Road frame $(X_r, Y_r, Z_r)^T$
- Road frame according to [ISO8855 norm](https://www.sis.se/api/document/preview/914200/) $(X_i, Y_i, Z_i)^T$

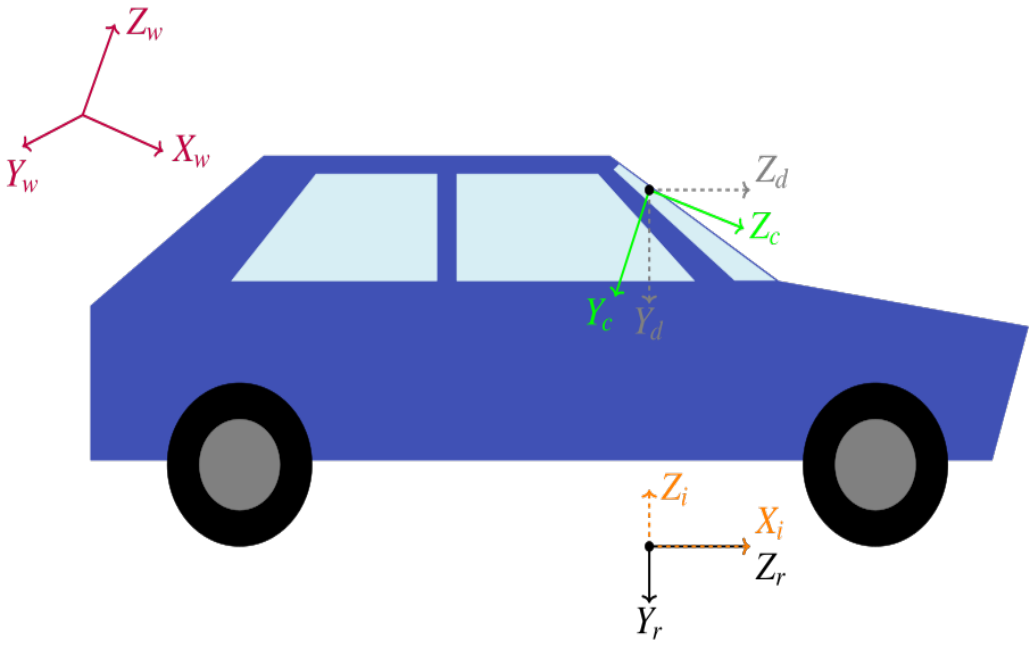

Figure 8: Reference frames

These frames are defined in the following figure. The axes of the default camera frame correspond to the ”right”, ”down”, and ”forwards” directions of the vehicle. Even if we wanted to mount our camera in a way that the *camera frame* equals the *default camera frame*, we might make some small errors. In the exercises, the *camera frame* is slightly rotated with respect to the *default camera frame*, but only along the X_d-axis: The camera has a of 5°.

In the following, we will describe how to transform between world and camera coordinates, but the same mathematics hold for a transformation between any two of the above reference frames.

A point $(X_w, Y_w, Z_w)^T$ in the world coordinate system is mapped to a point in the camera coordinate system via a rotation matrix as below

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = \mathbf{R} \begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} + \mathbf{t}$$

We can write this transformation law just using matrix multiplication:

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} = \begin{pmatrix} R_{xx} & R_{xy} & R_{xz} & t_x \\ R_{yx} & R_{yy} & R_{yz} & t_y \\ R_{zx} & R_{zy} & R_{zz} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} = \mathbf{T}_{cw} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$$

where we defined the transformation matrix $\mathbf{T}_{cw}$ in the last equality. It is great that we can relate the coordinates of different reference frames just by using a matrix. This makes it easy to compose several transformations, for example, to go from world to camera coordinates and then from camera to road coordinates. It also enables us to get the inverse transformation using the matrix inverse: The transformation from camera to world coordinates is given by the matrix $\mathbf{T}_{wc} = \mathbf{T}_{cw}^{-1}$.

To conclude this section, we combine eq-intrinsic-matrix-multiplication and change-coordinate-systems into one single equation:

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{R}_{xx} & \mathbf{R}_{xy} & \mathbf{R}_{xz} & t_x \\ \mathbf{R}_{yx} & \mathbf{R}_{yy} & \mathbf{R}_{yz} & t_y \\ \mathbf{R}_{zx} & \mathbf{R}_{zy} & \mathbf{R}_{zz} & t_z \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$$

Or by defining the intrinsic camera matrix $\mathbf{K}$ and the extrinsic camera matrix $\big(\mathbf{R}|\mathbf{t}\big)$:

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathbf{K} \big(\mathbf{R}|\mathbf{t}\big) \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$$

Yaw, Pitch and roll rotations

It's crucial to grasp the concept of yaw, pitch, and roll parametrization of rotations. These terms describe the rotation between the camera frame and the default camera frame.

1. **Yaw:** Yaw refers to the rotation around the vertical axis, typically the y-axis. It's akin to turning your head from side to side.
2. **Pitch:** Pitch involves rotation around the lateral axis, often the x-axis. It's similar to nodding your head up and down.
3. **Roll:** Roll entails rotation around the longitudinal axis, usually the z-axis. It's akin to tilting your head from one side to the other.

By combining these rotations in a specific order, we can describe any arbitrary rotation between the camera frame and the default camera frame. First, we rotate by a certain angle (roll angle) around the z-axis, then by another angle (pitch angle) around the x-axis, and finally by another angle (yaw angle) around the y-axis. This sequence of rotations provides a comprehensive description of the orientation of the camera relative to the default camera frame.

If we define the cosine and sine of the roll angle as c_r and s_r respectively, the roll rotation matrix is

$$R_{roll} = \begin{pmatrix} c_r & -s_r & 0 \\ s_r & c_r & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Defining cosine and sine of the pitch angle as c_p and s_p and the cosine and sine of the yaw angle as c_y and s_y, the pitch and yaw rotation matrices are

$$R_{pitch} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_p & s_p \\ 0 & -s_p & c_p \end{pmatrix}$$

$$R_{yaw} = \begin{pmatrix} c_y & 0 & -s_y \\ 0 & 1 & 0 \\ s_y & 0 & c_y \end{pmatrix}$$

Note that the form of these matrices depends on the coordinate system. We chose the $(X_d, Y_d, Z_d)^T$ frame here. In the $(X_i, Y_i, Z_i)^T$ frame the matrices would look a bit different, since for example the "forwards" direction in the $(X_i, Y_i, Z_i)^T$ frame is $(1, 0, 0)^T$, whereas it is $(0, 0, 1)^T$ in the default camera frame $(X_d, Y_d, Z_d)^T$. So if you find a differently looking roll matrix in some book, this probably has to do with the naming of the axes. What all books tend to agree on: In the case of vehicles (like cars or planes), the roll axis points forwards, the yaw axis upwards (or downwards), and the pitch axis to the right (or left).

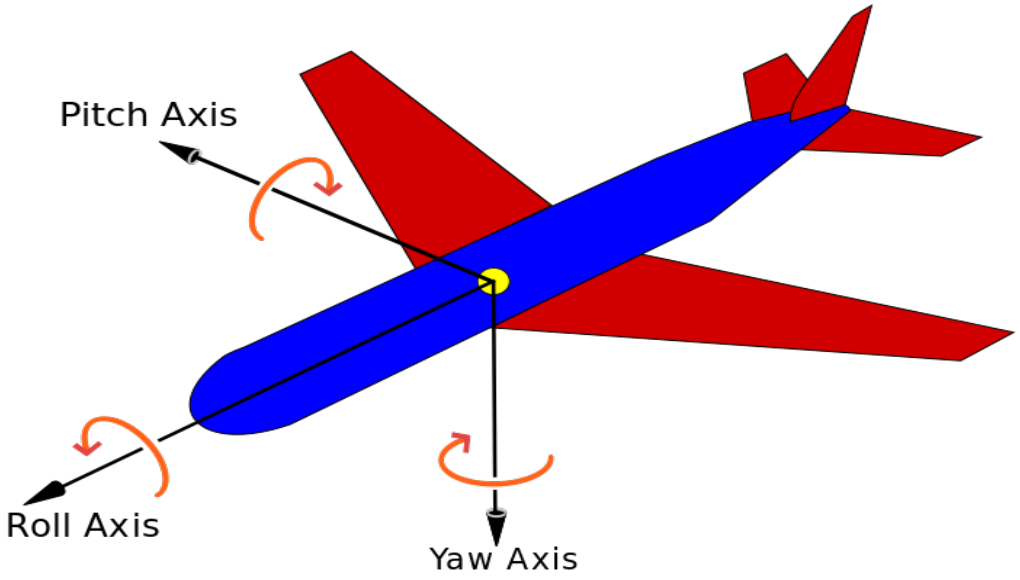

Figure 9: Yaw, pitch and roll rotations

A general rotation can be written by multiplying the roll, pitch, and yaw matrices

$$R = R_{yaw} R_{pitch} R_{roll} = \begin{pmatrix} c_r c_y + s_p s_r s_y & c_r s_p s_y - c_y s_r & -c_p s_y \\ c_p s_r & c_p c_r & s_p \\ c_r s_y - c_y s_p s_r & -c_r c_y s_p - s_r s_y & c_p c_y \end{pmatrix}$$

Converting pixels to meters

Having detected which pixel coordinates (u, v) are part of a lane boundary, we now want to determine which 3-dimensional points $(X_c, Y_c, Z_c)^T$ correspond to these pixel coordinates (u, v). Let's revisit the sketch of the image formation process:

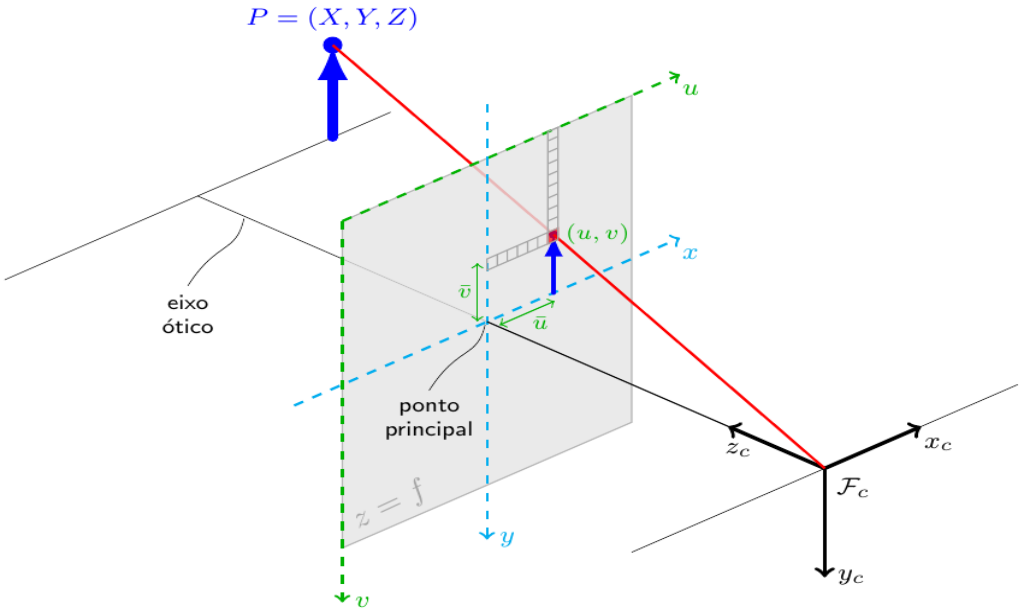

Figure 10: Image formation

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathbf{K} \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix}$$

However, now we need to solve the inverse problem. Given (u, v), we need to find $(X_c, Y_c, Z_c)^T$. To do this, we multiply the above equation by $\mathbf{K}^{-1}$:

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = \lambda \mathbf{K}^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

The issue is that we don't know the value of λ. Thus, the 3D point $(X_c, Y_c, Z_c)^T$ corresponding to pixel coordinates (u, v) lies somewhere on the line defined by:

$$\mathbf{r}(\lambda) = \lambda \mathbf{K}^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}, \quad \lambda \in R_0$$

However, determining which λ yields the point captured in our image is generally challenging. Here, we exploit our knowledge that $\mathbf{r}(\lambda)$ should lie on the road, as it corresponds to a point on the lane boundary. Assuming the road is planar, it can be characterized by a normal vector $\mathbf{n}$ and a point lying on the plane $\mathbf{r}_0$:

$$\text{Point } \mathbf{r} \text{ lies in the plane} \quad \Leftrightarrow \quad \mathbf{n}^T(\mathbf{r} - \mathbf{r}_0) = 0$$

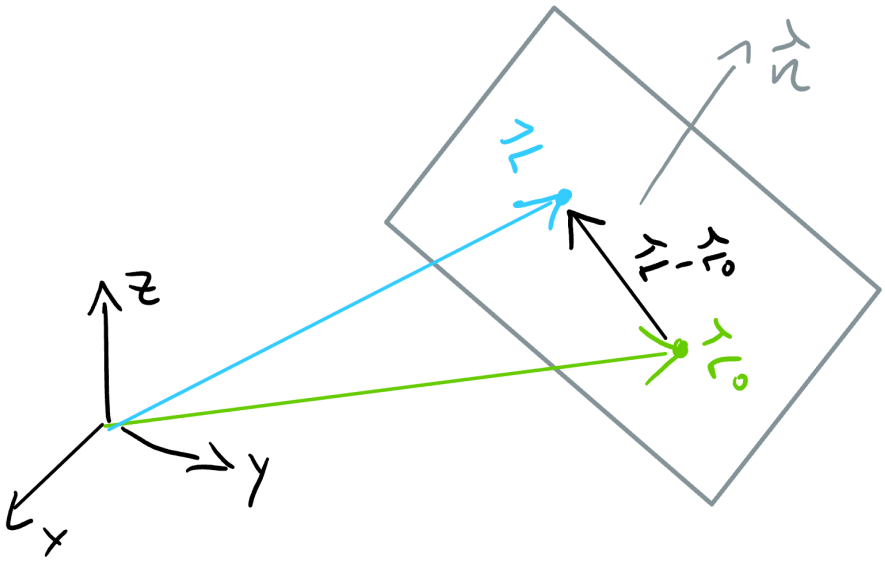

Figure 11: Image of a planar surface

In the road reference frame the normal vector is just $\mathbf{n} = (0, 1, 0)^T$. Since the optical axis of the camera is not parallel to the road, the normal vector in the camera reference frame is $\mathbf{n_c} = \mathbf{R_{cr}}(0, 1, 0)^T$, where the rotation matrix $\mathbf{R_{cr}}$ describes how the camera is oriented with respect to the road: It rotates vectors from the road frame into the camera frame. The remaining missing piece is some point $\mathbf{r}_0$ on the plane. In the camera reference frame, the camera is at position $(0, 0, 0)^T$. If we denote the height of the camera above the road by h, then we can construct a point on the road by moving from $(0, 0, 0)^T$ in the direction of the road normal vector $\mathbf{n_c}$ by a distance of h: Hence, we pick $\mathbf{r}_0 = h\mathbf{n_c}$, and our equation for the plane becomes $0 = \mathbf{n_c}^T(\mathbf{r} - \mathbf{r}_0) = \mathbf{n_c}^T\mathbf{r} - h$ or $h = \mathbf{n_c}^T\mathbf{r}$.

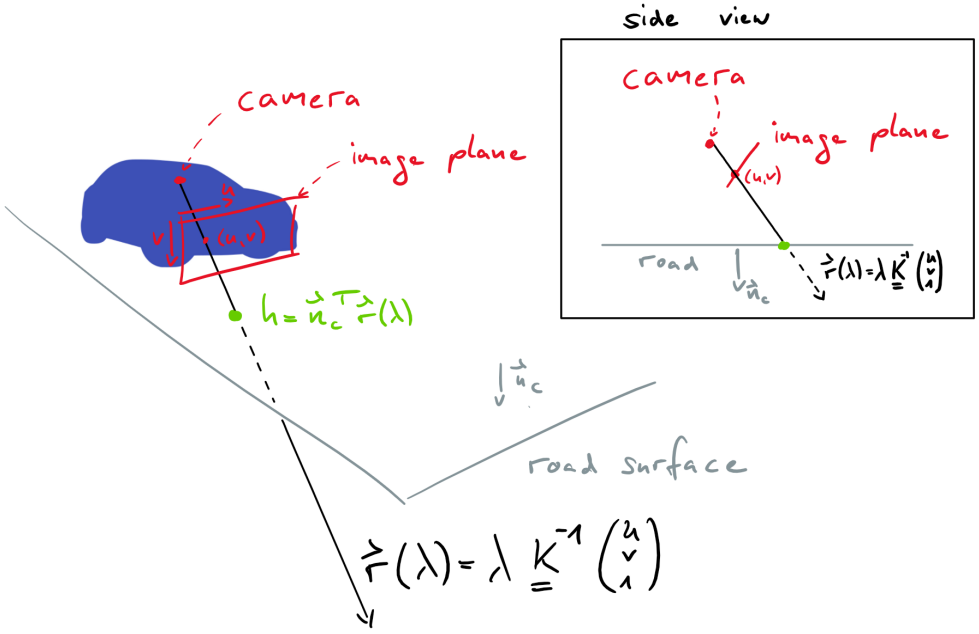

Figure 12: Finding the correct λ

Now we can compute the point where the line $\mathbf{r}(\lambda) = \lambda \mathbf{K}^{-1}(u, v, 1)^T$ hits the road, by plugging $\mathbf{r}(\lambda)$ into the equation of the plane $h = {\mathbf{n_c}}^T \mathbf{r}$

$$h = {\mathbf{n_c}}^T \lambda \mathbf{K}^{-1}(u, v, 1)^T \Leftrightarrow \lambda = \frac{h}{{\mathbf{n_c}}^T \mathbf{K}^{-1}(u, v, 1)^T}$$

We can now plug this value of λ into $\mathbf{r}(\lambda)$ to obtain the desired mapping from pixel coordinates (u, v) to 3 dimensional coordinates in the camera reference frame

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = \frac{h}{{\mathbf{n_c}}^T \mathbf{K}^{-1}(u, v, 1)^T} \mathbf{K}^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

Further, we implement all these formulas in Python and estimate the depth from the lanes predicted. The code can be found in the footnote[1].

Depth estimation using Inverse perspective mapping

Further, utilizing the aforementioned code, we estimate the depth of several points on the ground, primarily within a range of $\pm$ 40 cm width and up to a depth of 100 cm. Subsequently, we present the chart illustrating the calculation error between the actual and estimated depth values.

The below is the experimental setup for the reference.

[1] link to code: here

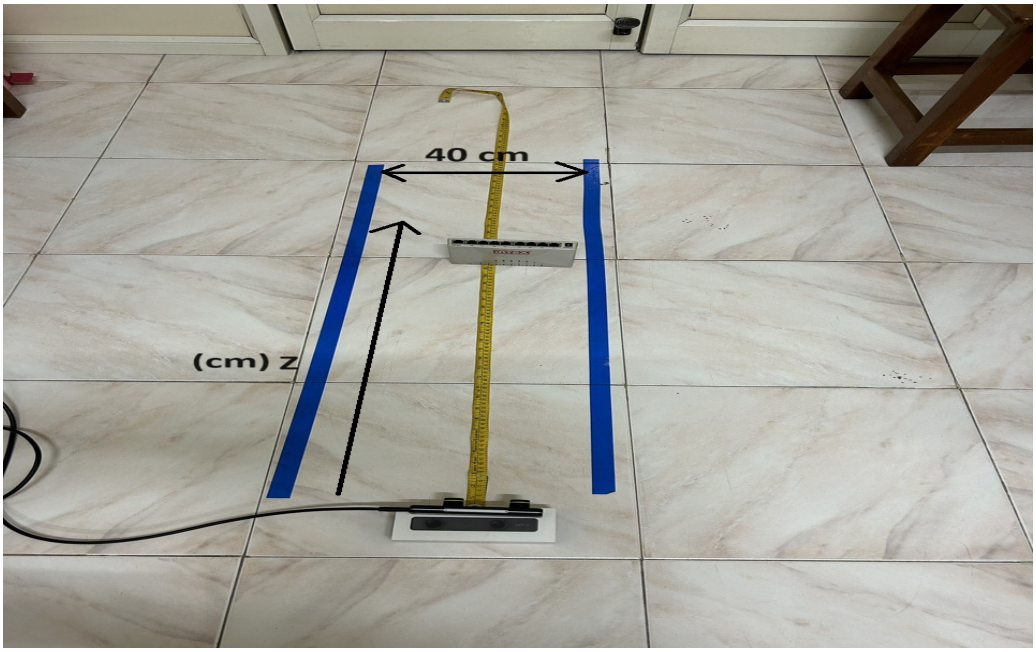

Figure 13: Experimental setup

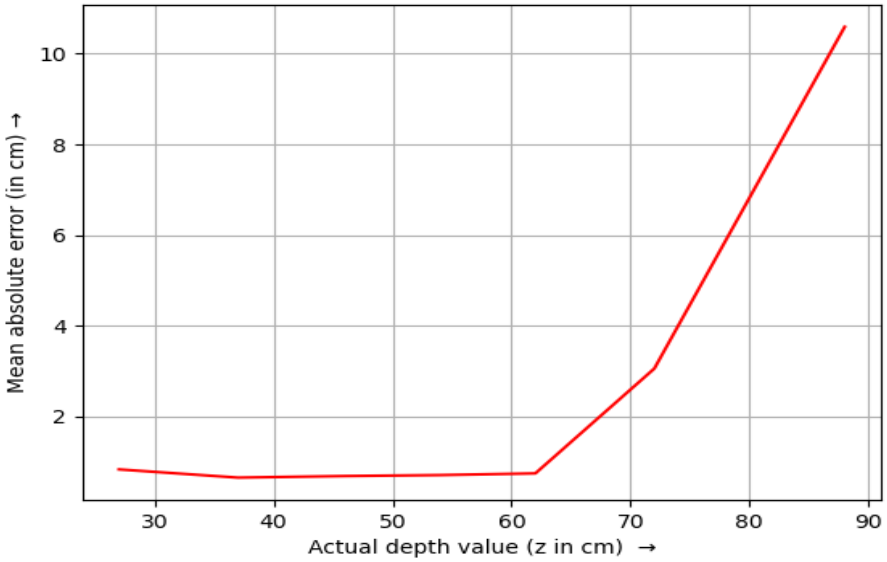

Figure 14: Mean absolute error between the actual and estimated depth values

It's evident from the figure above that the method performs admirably up to a depth of 70 cm, with errors averaging less than $\pm$ 3 cm. This indicates its suitability for depth estimation purposes. The data for the above experiment can be found in the footnote[2]

[2]link to code: here

Depth estimation using zed for objects

Here, we explore how the built-in point cloud methods of the ZED camera can be employed for object detection and depth estimation. The figure below illustrates the mean absolute error in depth estimation compared to the actual depth for an object (router here), utilizing the same experimental setup as described previously.

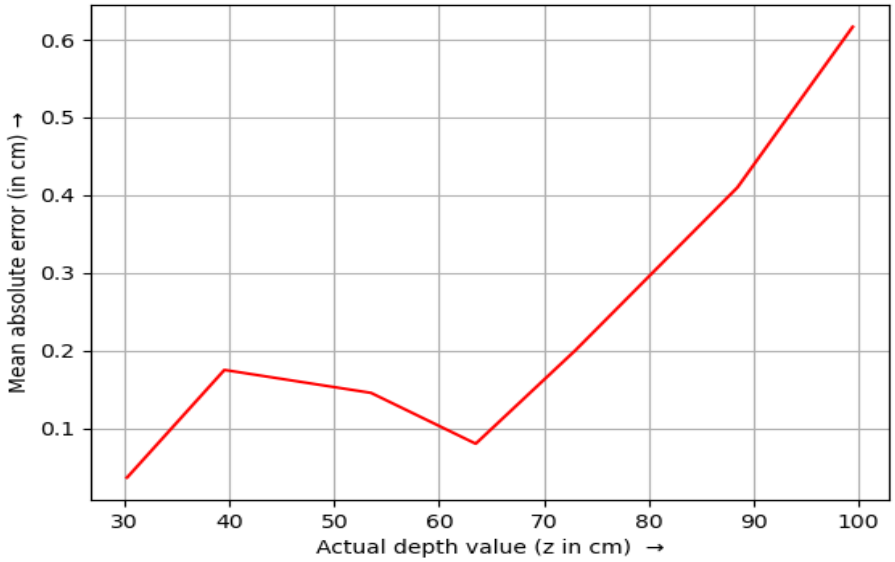

Figure 15: Mean absolute error between the actual and estimated depth values

It is evident that both methods provide accurate depth estimates within our range of interest. Therefore, we can confidently integrate these approaches into our path planning algorithms. The link to the data [3] and the code [4] is given in the footnote.

Path planning algorithm

Utilizing the aforementioned depth estimation and segmentation methods, we implement a straightforward yet efficient path planning and waypoint calculation algorithm. The algorithm comprises the following steps:

1. Capture an image using the ZED camera.
2. Transmit the captured image to the server computer.
3. Perform image segmentation to identify drivable area.
4. Extract extreme points corresponding to white pixels along a given horizontal line in the image.
5. Iterate this process across the entire image to obtain multiple sets of extreme points.

[3] link to data: here
[4] link to code: here

6. Compute the average pixel values for each set of extreme points and determine the midpoints along the vertical axis.
7. Estimate the corresponding x and z values for these midpoints, which serve as the waypoints.
8. Transmit these waypoints back to the vehicle and navigate it to follow this path.
9. Repeat the process iteratively.

The following flowchart illustrates the sequence of steps involved in this algorithm.

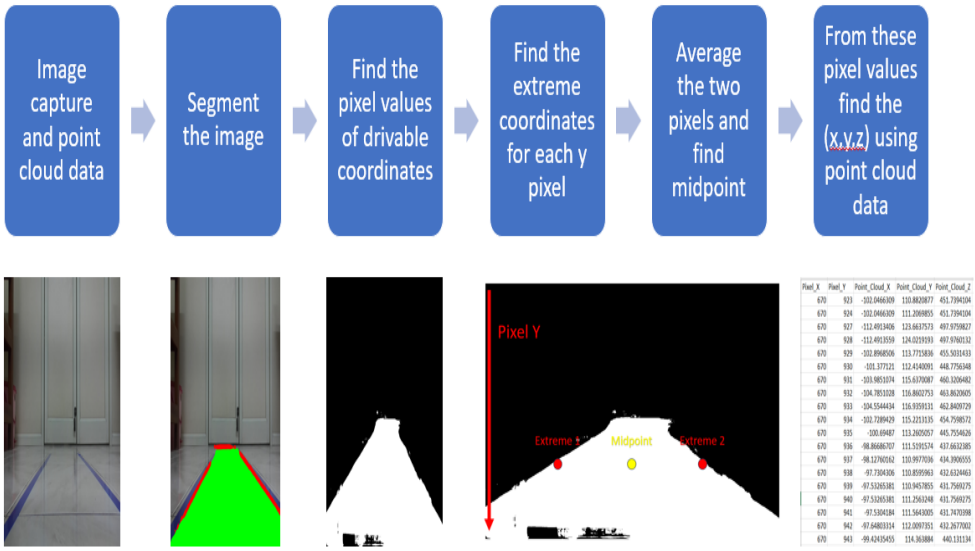

Figure 16: Flow chart

Experiment

Further, we plan to conduct experiments to validate the effectiveness of the developed algorithms on a vehicle in a laboratory setting. The experiment entails configuring a curved track and guiding our vehicle along the designated lanes.

The track curvature is as depicted in the image below:

Figure 17: Track

Thank You