Software module for autonomous car

Student: Rahil Amit Shah
Roll Number: EE20B104

B.Tech project report

Guide: Dr. Ramkrishna Pasumarthy

Dept: Electrical Engineering
IIT Madras
India
Date: May 22, 2024

Abstract

The rapid evolution of autonomous vehicle technology has underscored the necessity for robust software modules that ensure safe and efficient navigation. This project aims to meet this demand by developing and integrating a sophisticated software module tailored specifically for autonomous cars of one-tenth scale. Our methodology comprises multiple stages, beginning with the utilization of the YOLOP model for scene segmentation to identify drivable areas. Subsequent stages involve generating waypoints for path planning and estimating 3D coordinates to guide vehicle navigation. A crucial aspect of this endeavor involves exploring and experimenting with various depth estimation methods to enhance accuracy and reliability.

Moreover, this project contributes to the ongoing discourse on autonomous vehicle technology by exploring novel approaches to address critical challenges, specifically tailored for a one-tenth scale car. The integration of state-of-the-art algorithms and techniques, coupled with rigorous experimentation and validation, underscores our commitment to advancing knowledge in this domain. Furthermore, the insights gleaned from this research have broader implications for the broader field of artificial intelligence and robotics, paving the way for the development of more sophisticated and adaptive autonomous systems in the future.

Additionally, scaled-down vehicles offer a controlled testing environment for autonomous driving algorithms. They play a pivotal role in studying the impact of V2V (vehicle-to-vehicle) and V2I (vehicle-to-infrastructure) communication on cooperative autonomous vehicle behavior, traffic efficiency, and road safety within confined spaces. This controlled experimentation expedites algorithm validation and development while minimizing risks associated with full-scale testing. We introduce a meticulously designed one-tenth-scale electric vehicle, emphasizing precise vehicle state measurement, control, perception sensing, and electric safety.

Finally, this project serves as a testament to the potential of collaborative research efforts and interdisciplinary collaboration in driving innovation and advancing technology in autonomous vehicle systems. Through meticulous experimentation and validation, coupled with a commitment to pushing the boundaries of knowledge in this domain, we aim to contribute to the continued evolution of autonomous vehicle technology and its broader applications in society.

Contents

Acknowledgments

The successful completion of this project owes much to the guidance and support of several individuals, without whom this endeavor would not have been possible.

I extend my heartfelt gratitude to **Dr. Ramkrishna Pasumarthy** and **Dr. Nirav Bhatt**, whose mentorship played a pivotal role in steering the project towards success. Their profound insights, constructive feedback, and unwavering encouragement throughout every stage of the project were invaluable. Their expertise in the field of autonomous vehicles and software development provided a solid foundation upon which this research was built.

Special thanks are due to **Subhadeep sir**, whose tireless dedication, sage advice, and mentorship throughout the entire project were instrumental. His profound understanding of complex technical concepts and his ability to provide practical solutions to challenges encountered during the project were truly commendable. His guidance not only enriched the project but also fostered personal and professional growth.

I would also like to express my appreciation to **Soumyajit sir**, whose continuous guidance and deep knowledge in the fields of deep learning and computer vision served as a solid foundation for this project. His insightful discussions, scholarly expertise, and willingness to share his knowledge were invaluable assets. His mentorship not only enhanced the technical aspects of the project but also broadened my understanding of the subject matter.

I am grateful to **Vasumathi ma'am**, for her indispensable assistance in setting up experiments. Her precise understanding of the dynamics of car motion and track conditions proved instrumental in conducting experiments successfully. Her expertise ensured that the experimental setup was meticulously designed, and data collection was carried out with precision.

Collectively, the contributions of these individuals have been instrumental in the successful execution of this project, and I extend my sincere thanks for their unwavering support, expertise, and guidance.

Introduction

The emergence of autonomous vehicles marks a pivotal moment in transportation, heralding a transformative shift towards safer, more efficient, and eco-friendly modes of mobility compared to conventional human-operated vehicles. At the heart of achieving autonomous driving lies the development of sophisticated software modules. These modules empower vehicles to perceive their surroundings, chart optimal routes, and navigate autonomously through intricate and ever-changing landscapes.

This project is dedicated to crafting and seamlessly integrating a bespoke software module tailored expressly for autonomous cars of one-tenth scale, with the overarching goal of enriching their navigation capabilities. Encompassing a suite of critical functionalities such as scene segmentation, path planning, waypoint derivation, and depth estimation, this module draws upon cutting-edge algorithms and methodologies drawn from the intersecting realms of computer vision, deep learning, and robotics. Through these innovative techniques, our software enables autonomous vehicles to comprehensively comprehend their environment, make informed decisions, and adeptly traverse towards their destinations with safety and efficiency as top priorities.

Commencing with the utilization of the YOLOP model, our project initiates the process of scene segmentation, empowering vehicles to discern and demarcate drivable areas within their surroundings. Subsequent stages involve waypoint derivation, essential for charting a secure and optimal trajectory. Augmenting these waypoints with 3D coordinates further bolsters navigation precision, enabling vehicles to navigate through intricate terrains and circumvent obstacles with unwavering accuracy.

A pivotal phase of our project revolves around the exploration and experimentation of diverse depth estimation methods. By subjecting these methods to rigorous evaluation under varying environmental conditions, our aim is to ascertain the most viable approach for real-world deployment. These experiments serve as a crucial litmus test, ensuring that our navigation system is both reliable and resilient.

Validation of our software module's efficacy takes place through meticulously designed laboratory experiments, where the vehicle's movements prompt real-time estimation of waypoints. Moreover, we develop a comprehensive probabilistic model to assess the error associated with waypoint calculation, thereby providing invaluable insights into the robustness and reliability of our navigation system.

The illustration below encapsulates the project's workflow:

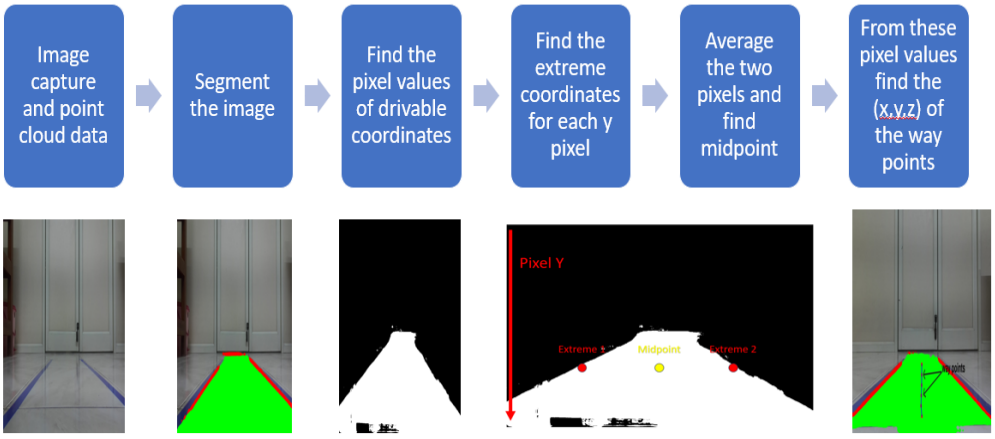


Figure 1: Flow chart

In essence, our project represents a significant stride towards advancing autonomous vehicle technology. By furnishing a comprehensive software solution that amplifies the navigation prowess of autonomous cars, we inch closer towards realizing fully autonomous transportation systems. Through the amalgamation of state-of-the-art algorithms and techniques, our software module empowers autonomous vehicles to navigate with confidence and precision across diverse landscapes, thus laying the groundwork for a future defined by seamless and safe autonomous mobility.

YOLOP

YOLOP Architecture

The YOLOP (YOLOP: You Only Look Once for Panoptic Driving Perception) architecture is an advanced variant of the popular YOLO (You Only Look Once) object detection framework. YOLOP builds upon the strengths of YOLO while introducing innovative techniques to improve efficiency and reduce computational complexity.

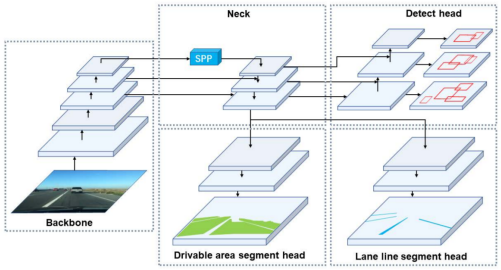


Figure 2: YOLOP Architecture

YOLOP has one shared encoder and three decoder heads to solve specific tasks. There are no complex shared blocks between different decoders to keep the computation to a minimum and allow for easier end-to-end training.

Encoder

The encoder in YOLOP consists of a backbone network and a neck network. The backbone network, CSP-Darknet, extracts features from input images. It leverages feature reuse and propagation to reduce parameters and computations. The neck network performs feature engineering, combining the Spatial Pyramid Pooling (SPP) module and the Feature Pyramid Network (FPN) module. These modules generate and fuse features of different scales and semantic levels, resulting in rich features with multi-scale and multi-level information.

Decoders

YOLOP uses different decoders for various tasks. For object detection, it adopts an anchor-based multi-scale detection technique similar to YOLOv4. The detection head, Path Aggregation Network (PAN), combines semantic features from the FPN top-down and image features from the PAN bottom-up to improve feature fusion. This fusion creates a multi-scale feature map used for object detection.

For drivable area segmentation and lane line segmentation, YOLOP utilizes the same network structure. The features from the bottom layer of the FPN, with size (W/8, H/8, 256), are fed into the segmentation branch. Through three upsampling processes, the feature map is restored to (W, H, 2), representing the pixel-wise probability for drivable areas and lane lines. YOLOP doesn't require an additional SPP module for segmentation heads due to the shared SPP module in the neck network.

By employing this architecture, YOLOP effectively extracts features from images and performs tasks such as object detection, drivable area segmentation, and lane line segmentation.

Data

The BDD100K dataset, specifically designed for autonomous driving applications, has played a vital role in training and evaluating our YOLOP-based segmentation module. It consists of over 100,000 high-resolution images captured across diverse driving scenarios, encompassing various driving conditions, weather patterns, and lighting scenarios.

The dataset provides comprehensive annotations for object detection, lane detection, and semantic segmentation tasks. These annotations, generated through a combination of manual segmentation and automated algorithms, include pixel-level delineation of objects such as cars, pedestrians, cyclists, and traffic signs. Moreover, lane detection annotations facilitate accurate identification and tracking of lane markings, crucial for effective lane keeping and path planning.

Leveraging the rich annotations of the BDD100K dataset, our segmentation module underwent extensive training to simultaneously perform object detection, lane detection, and segmentation tasks. This training enabled the module to robustly analyze real-world driving scenes and deliver accurate and reliable segmentation results for autonomous driving applications. By utilizing the diverse and comprehensive BDD100K dataset, our segmentation module is well-equipped to handle the challenges encountered in complex driving environments, contributing to improved safety and performance in autonomous vehicles.

Evaluation of the model

To evaluate the performance of the YOLOP model, a comprehensive metric is required that takes into account the different tasks it performs. Traditional evaluation metrics may not be suitable as they focus on individual tasks and fail to capture the overall performance of the multitask model. Therefore, a specialized loss function is introduced in the paper to effectively measure the performance of YOLOP.

By utilizing this specialized loss function, YOLOP can effectively evaluate the model's performance across multiple tasks and ensure optimal performance for the panoptic driving perception application.

Losses

The YOLOP architecture incorporates several loss functions to effectively train the model for different tasks. These losses are designed to capture specific aspects of the model's performance. Here are the definitions of the losses used in YOLOP:

Detection Loss (L_{det})

The detection loss is a weighted sum of three components: the classification loss (L_{class}), the object loss (L_{obj}), and the bounding box loss (L_{box}). The classification loss penalizes misclassification, the object loss penalizes confidence in predictions, and the bounding box loss considers the similarity between predicted and ground truth bounding boxes.

Drivable Area Segmentation Loss ($L_{\text{da-seg}}$)

The drivable area segmentation loss is defined as the Cross Entropy Loss with Logits (L_{ce}). It aims to minimize the classification errors between the network outputs and the ground truth for drivable areas.

Lane Line Segmentation Loss ($L_{\text{ll-seg}}$)

The lane line segmentation loss combines the Cross Entropy Loss with Logits (L_{ce}) and the Intersection over Union Loss (L_{IoU}). The Cross Entropy Loss minimizes classification errors, and the Intersection over Union Loss is especially efficient for the prediction of sparse categories such as lane lines.

Overall Loss (L_{all})

The overall loss is a weighted sum of the detection loss, drivable area segmentation loss, and lane line segmentation loss. It is defined as $L_{\text{all}} = \gamma_1 L_{\text{det}} + \gamma_2 L_{\text{da-seg}} + \gamma_3 L_{\text{ll-seg}}$, where $\gamma_1, \gamma_2, \gamma_3$ are weights that can be adjusted to balance the different components of the total loss.

By optimizing these loss functions during training, YOLOP aims to achieve accurate and reliable performance across multiple tasks.

Multi-task Loss

Since YOLOP has three decoders, the multi-task loss consists of three parts. The detection loss, denoted as L_{det}, is a weighted sum of the classification loss (L_{class}), object loss (L_{obj}), and bounding box loss (L_{box}), as shown in Equation (1):

$$L_{\text{det}} = \alpha_1 L_{\text{class}} + \alpha_2 L_{\text{obj}} + \alpha_3 L_{\text{box}} \tag{1}$$

Here, L_{class} and L_{obj} are focal losses [12], which penalize well-classified examples to focus on harder ones. L_{box} is LCIoU [31], which considers distance, overlap rate, and similarity of scale and aspect ratio between the predicted box and the ground truth.

The drivable area segmentation loss ($L_{\text{da-seg}}$) and lane line segmentation loss ($L_{\text{ll-seg}}$) both utilize the Cross Entropy Loss with Logits (L_{ce}) to minimize classification errors between network outputs and targets. Additionally, $L_{\text{ll-seg}}$ includes the Intersection over Union (IoU) loss (L_{IoU}) to efficiently predict the sparse category of lane lines. The definitions for $L_{\text{da-seg}}$ and $L_{\text{ll-seg}}$ are given by Equations (2) and (3), respectively:

$$L_{\text{da-seg}} = L_{\text{ce}} \tag{2}$$

$$L_{\text{ll-seg}} = L_{\text{ce}} + L_{\text{IoU}} \tag{3}$$

In conclusion, the final loss (L_{all}) in YOLOP is a weighted sum of the three parts, as shown in Equation (4):

$$L_{\text{all}} = \gamma_1 L_{\text{det}} + \gamma_2 L_{\text{da-seg}} + \gamma_3 L_{\text{ll-seg}} \tag{4}$$

The values of $\alpha_1, \alpha_2, \alpha_3, \gamma_1, \gamma_2, \gamma_3$ can be tuned to balance the different components of the total loss.

Results of the YOLOP model

The inference time for each frame was 0.5729 seconds, while the non-maximum suppression (NMS) step took an additional 0.2138 seconds per frame.

It is important to note that these results were obtained using our current hardware setup, which has certain constraints. However, we believe that these performance metrics can be significantly improved by leveraging hardware accelerators specifically designed for deep learning tasks. By utilizing more powerful hardware resources, such as GPUs or specialized AI chips, we can enhance the speed and

Figure 3: Image 1

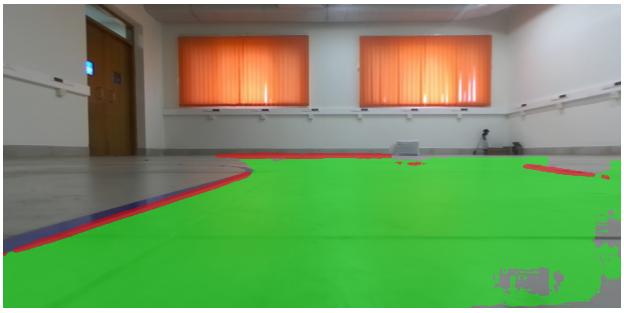


Figure 4: Prediction 1

Figure 5: Image 2

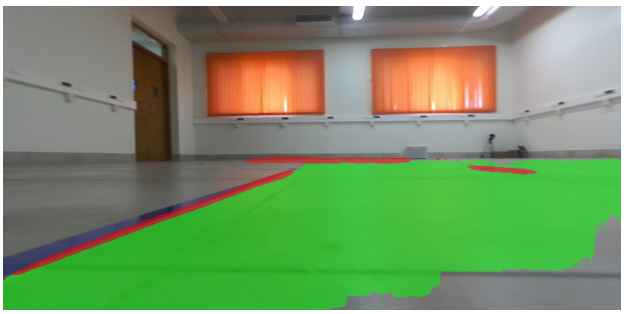


Figure 6: Prediction 2

Figure 7: Image 3

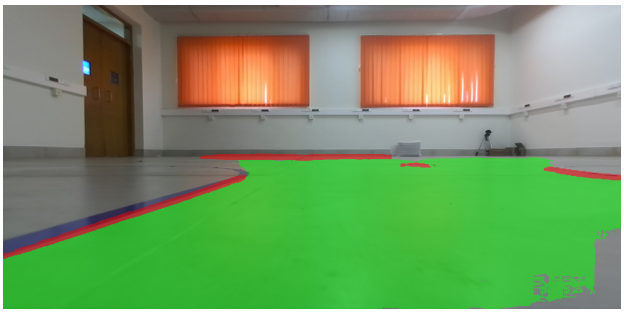


Figure 8: Prediction 3

Figure 9: Images and their predictions.

accuracy of our model, leading to better segmentation, lane detection, and object detection results.

Some of the images that we used to test the pre-trained model are as given below and more such videos and images can be found here[1].

The effectiveness of the implemented segmentation module and obstacle detection system is demonstrated through a series of result images captured during testing in the lab environment. These images provide visual insights into the model's performance in various scenarios, showcasing its ability to accurately detect lanes.

These result images validate the system's robustness and its ability to handle challenging conditions encountered in a controlled lab environment. Further in the next section we discuss about how we estimate the depth of these drivable coordinates.

[1] videos and images

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 10: 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 11: 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 12: 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 (reference[2]). 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.

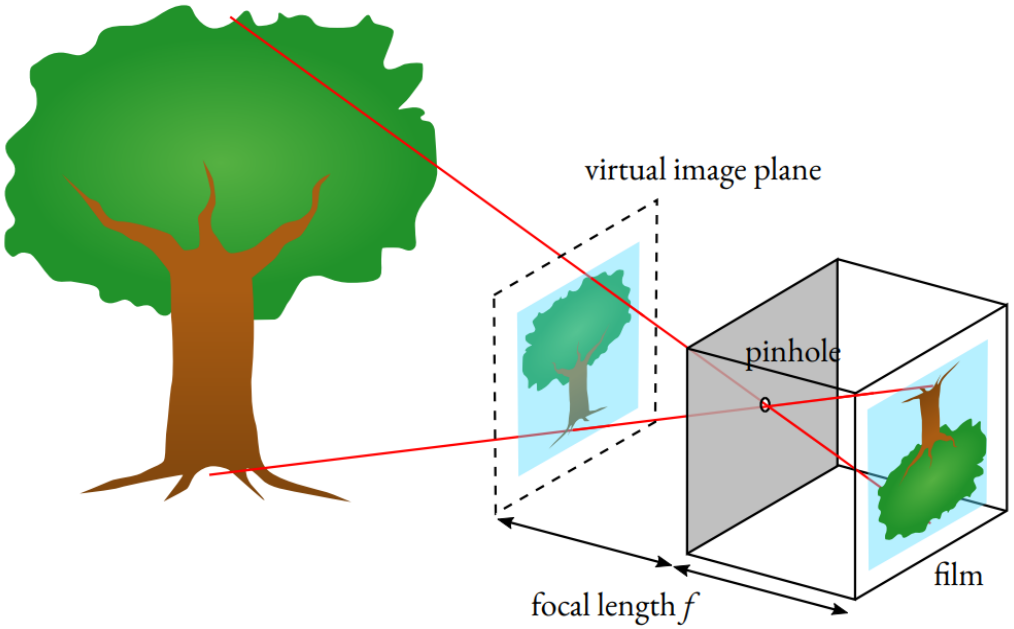


Figure 13: Pinhole camera

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.

[2]link to the research paper: here

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.

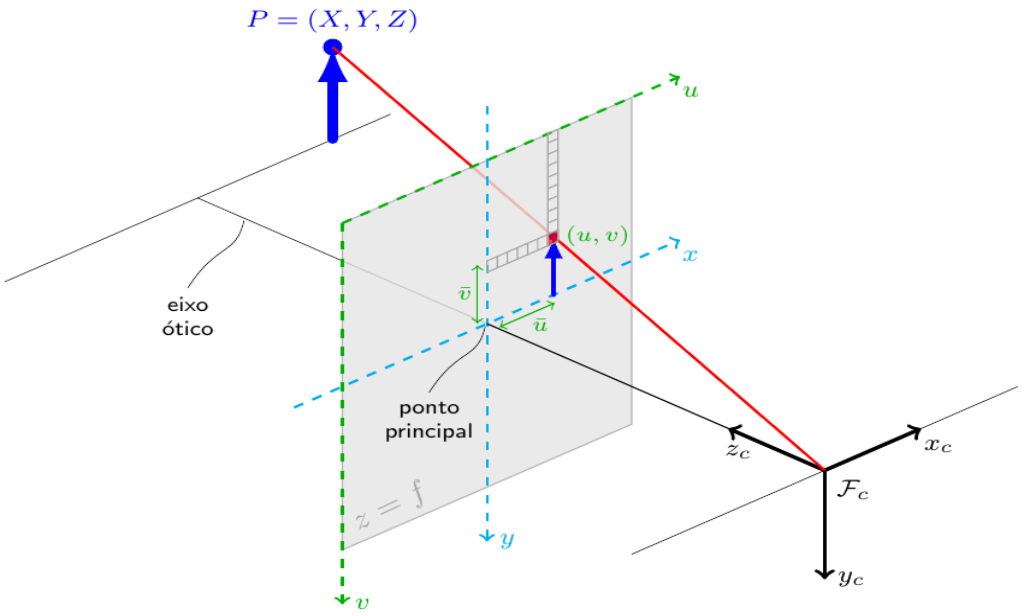


Figure 14: Image formation

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:

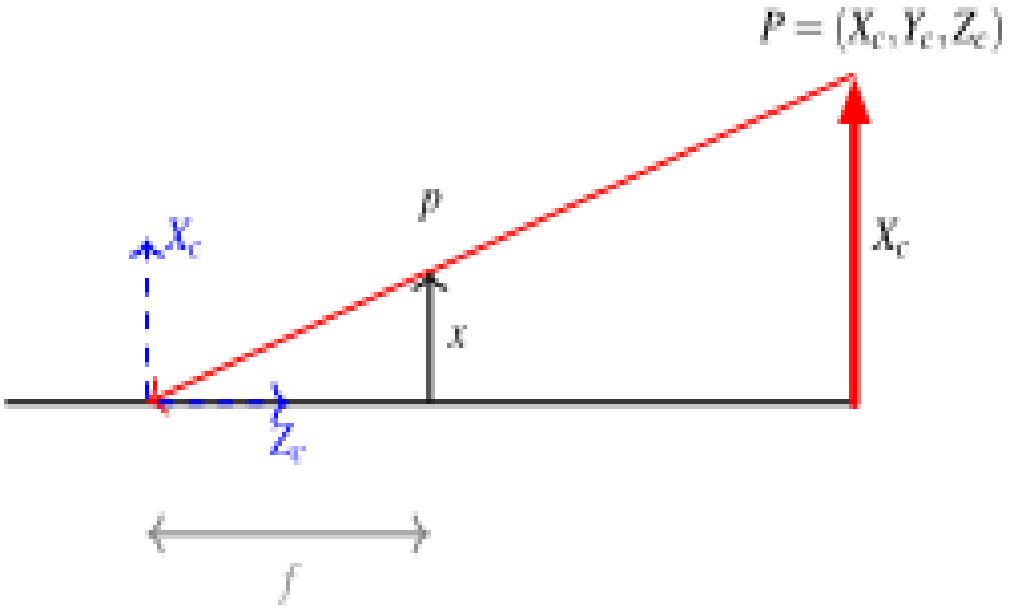


Figure 15: Pinhole camera

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) . 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/) $(X_i, Y_i, Z_i)^T$

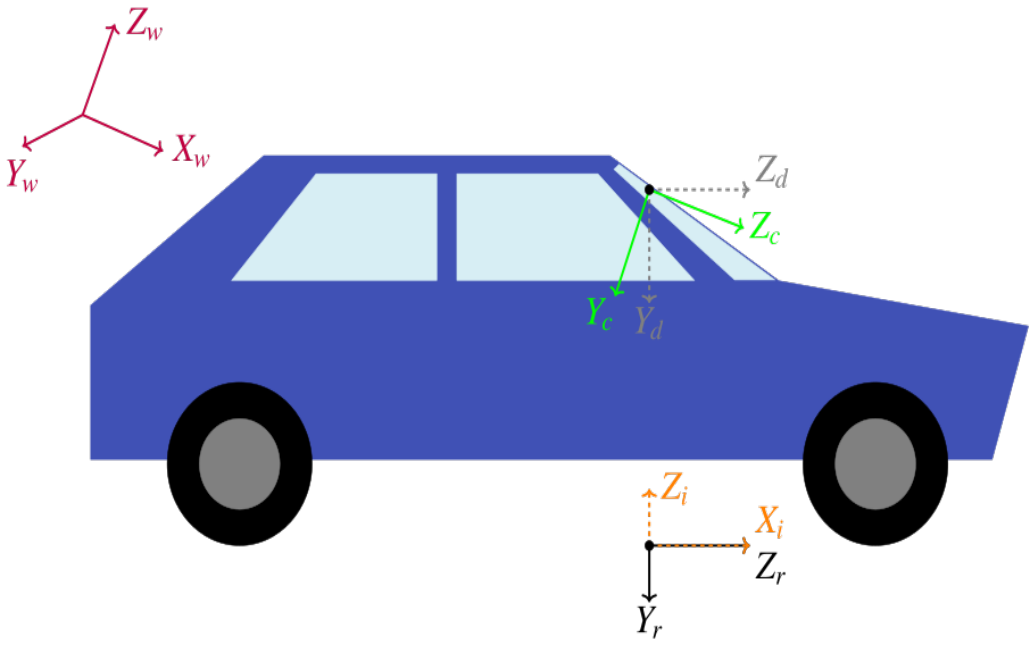


Figure 16: Reference frames

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 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

Understanding the concept of yaw, pitch, and roll parametrization of rotations is fundamental in various fields, particularly in applications involving spatial orientation and motion. These terms describe the rotational movements between a camera frame and a default reference frame.

1. **Yaw:** Yaw represents the rotation around the vertical axis, typically denoted as the y-axis. It can be visualized as the motion of turning one's head from side to side. In the context of a camera, yaw adjustment alters the horizontal orientation of the camera, allowing it to face different directions while maintaining its level orientation.

2. **Pitch:** Pitch involves rotation around the lateral axis, often represented by the x-axis. It resembles the motion of nodding one's head up and down. When applied to a camera, pitch adjustment changes the vertical angle of the camera, enabling it to capture views from varying heights or angles.

3. **Roll:** Roll refers to the rotation around the longitudinal axis, typically indicated as the z-axis. It can be visualized as tilting one's head from one side to the other. In the context of a camera, roll adjustment alters the horizon level of the camera, allowing it to compensate for tilts or banking motions and maintain a straight orientation relative to the ground.

These parametrizations play a crucial role in spatial orientation, navigation, and robotics, providing a standardized framework for describing and manipulating rotational movements in three-dimensional space.

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).

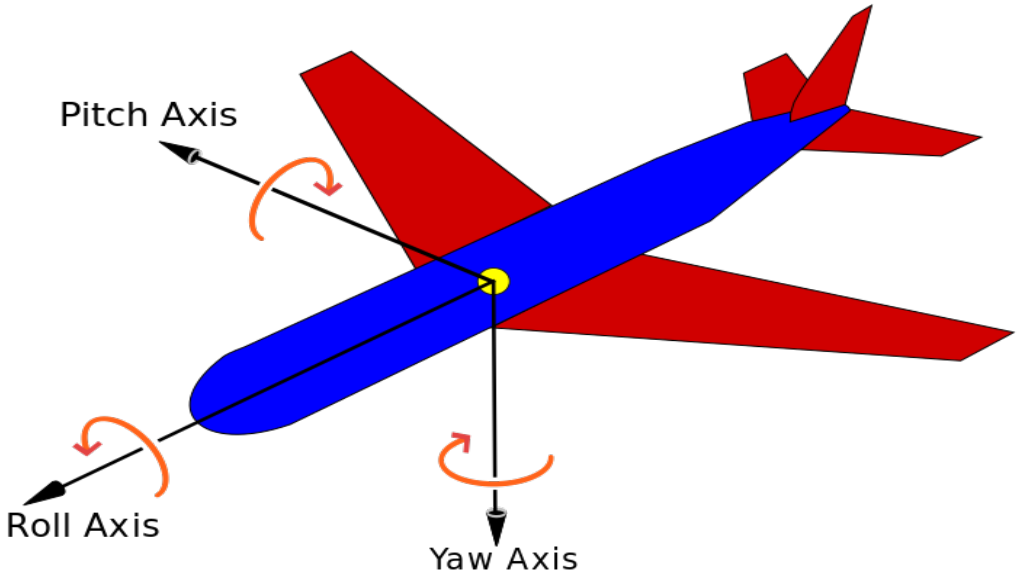


Figure 17: Yaw, pitch and roll rotations

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:

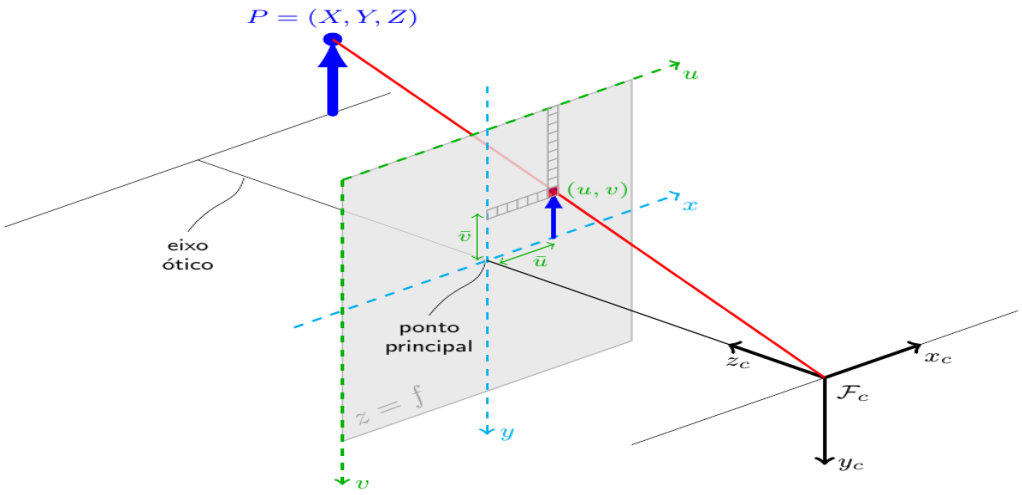


Figure 18: Image formation

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

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 $\mathbf{r}_0$:

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

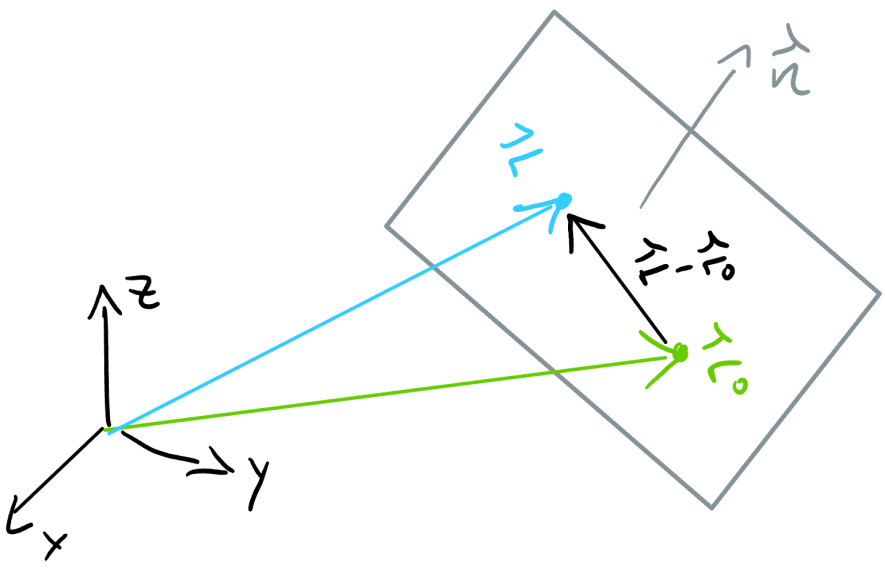


Figure 19: Image of a planar surface

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 $h = \mathbf{n_c}^T\mathbf{r}$.

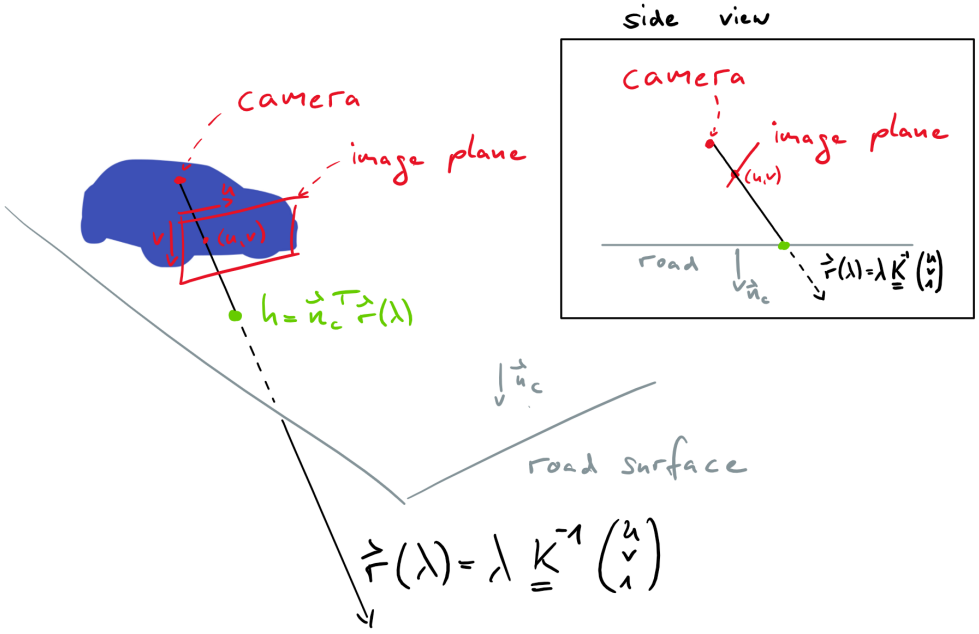


Figure 20: Finding the correct λ

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[3].

[3]link to code: here

Results of depth estimation

In this section, we present the outcomes of our experiments conducted for both depth estimation using inverse perspective mapping and ZED for object detection.

Depth Estimation using Inverse Perspective Mapping

In this section, we present the results obtained from our experiments on depth estimation using inverse perspective mapping (IPM). Our goal was to estimate the depth of various points on the ground within a range of $\pm$ 40 cm width and up to a depth of 100 cm. These estimates were then compared with ground truth values to assess the accuracy of our method.

The experimental setup involved placing a scale on the ground and measuring distances from the left camera, which served as the origin and the camera is placed at a height of 5.5 cm from the ground as in the vehicle. An object's bottom-most point was used as the reference for the points to be measured. The code used for these experiments can be found in the provided link in the footnote [4].

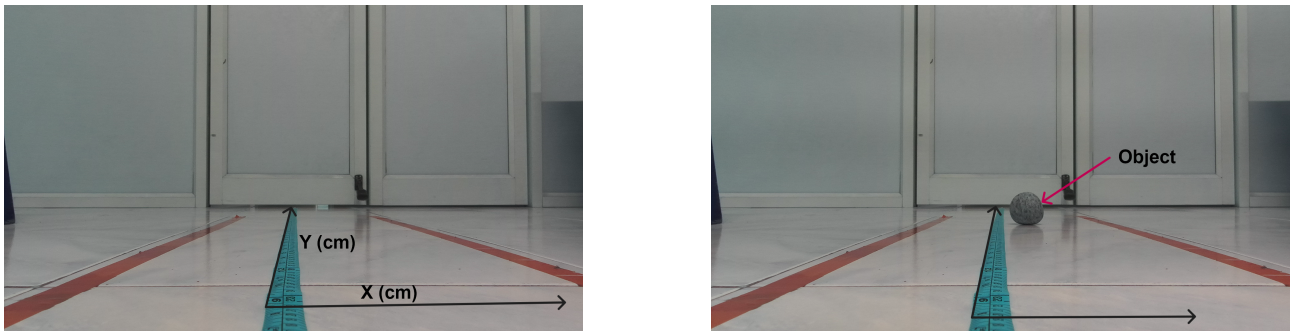


Figure 21: Experimental Setup

Images captured during the experiments are presented below, with the results compared against the actual ground truth.

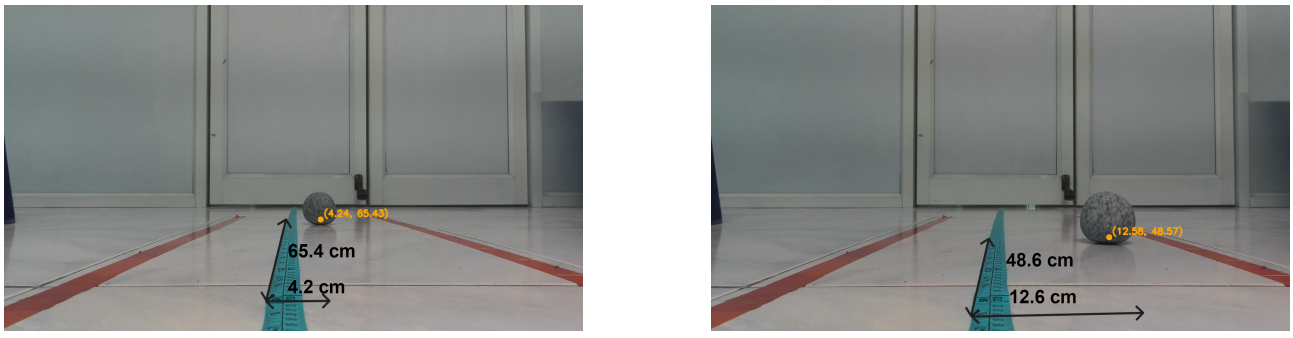


Figure 22: Experiment Results 1

[4] link to code: here

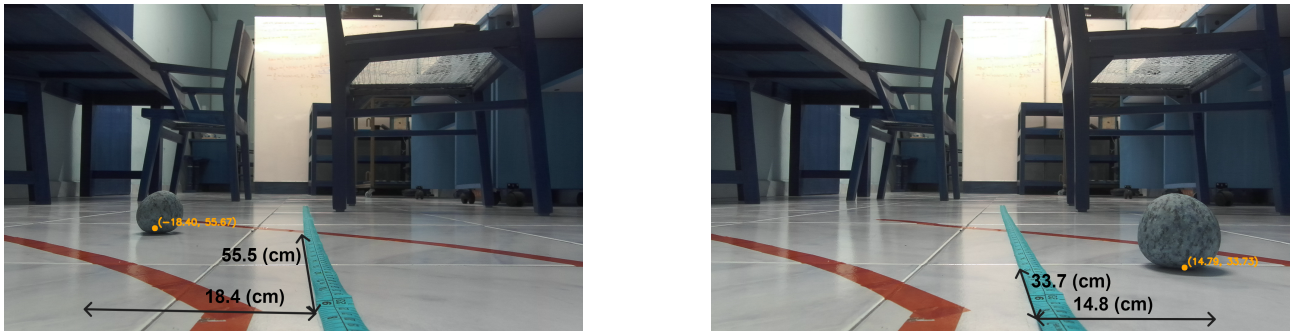


Figure 23: Experiment Results 2

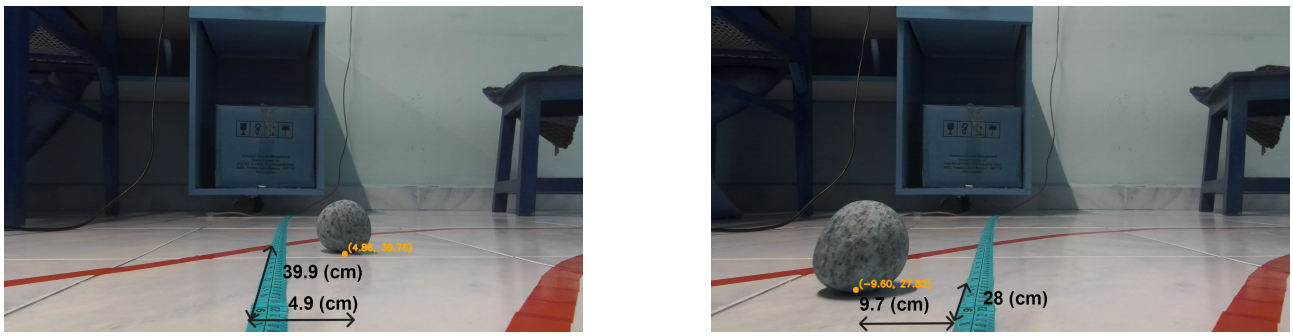


Figure 24: Experiment Results 3

Additionally, we conducted experiments for points ranging from 20 cm to 90 cm, with 10 cm intervals. Ten points were measured at each distance. The mean absolute error versus distance from the origin was then calculated and plotted.

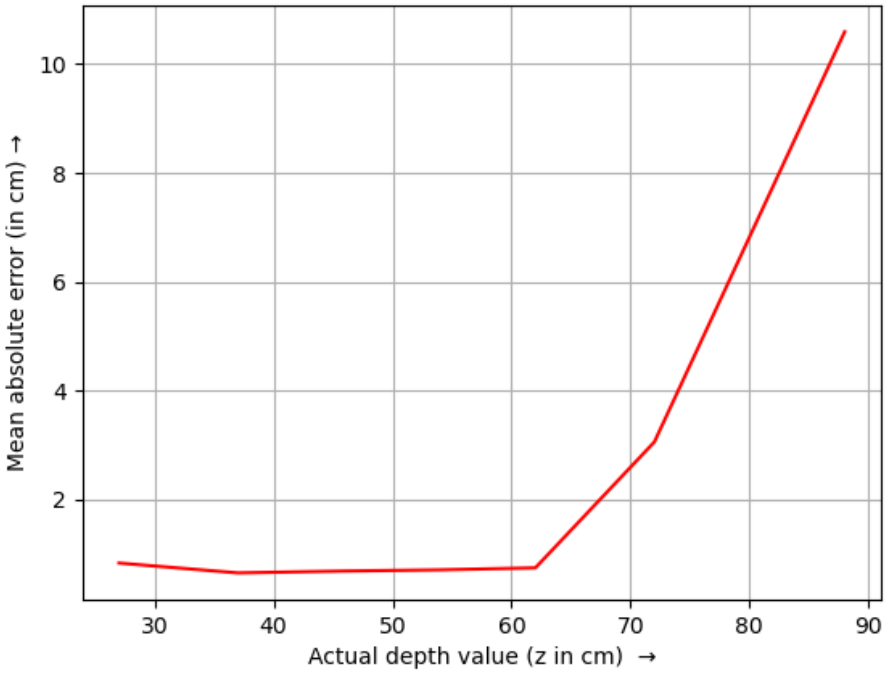


Figure 25: Mean Absolute Error vs. Distance

The plot illustrates that our method performs well up to a depth of 70 cm, with errors averaging less than $\pm$ 3 cm. Even at a depth of 90 cm, the error remains within $\pm$ 10 cm, highlighting the suitability of our approach for depth estimation. The experimental data can be found in the provided link in the footnote [5].

[5] link to data: here

Depth estimation using zed for objects

In this section, we investigate the utilization of the built-in point cloud methods of the ZED camera for object detection and depth estimation. The figure below demonstrates the mean absolute error in depth estimation compared to the actual depth for an object (a paperweight in this case). The experimental setup remains consistent with the previous description; however, in this scenario, we measure the depth of the object directly. Previously, the object served as a reference point, and we measured the distance of the point on the ground beneath the object.

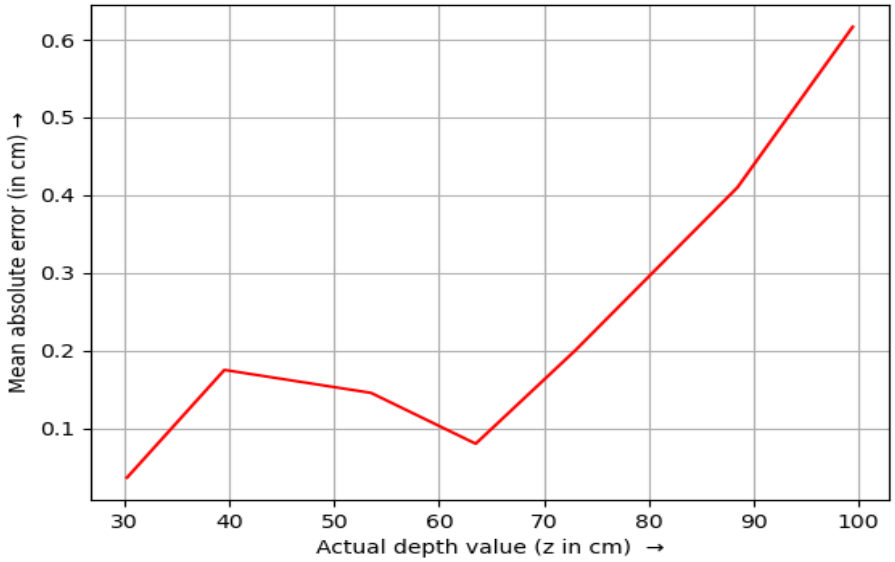


Figure 26: Mean absolute error between the actual and estimated depth values

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 [6] and the code [7] is given in the footnote.

[6] link to data: here
[7] link to code: here

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:

Algorithm 1 Path planning algorithm

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 using the two pointer approach.
5: Iterate this process across the entire image to obtain multiple sets of extreme points.
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 y 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 from step 1.

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

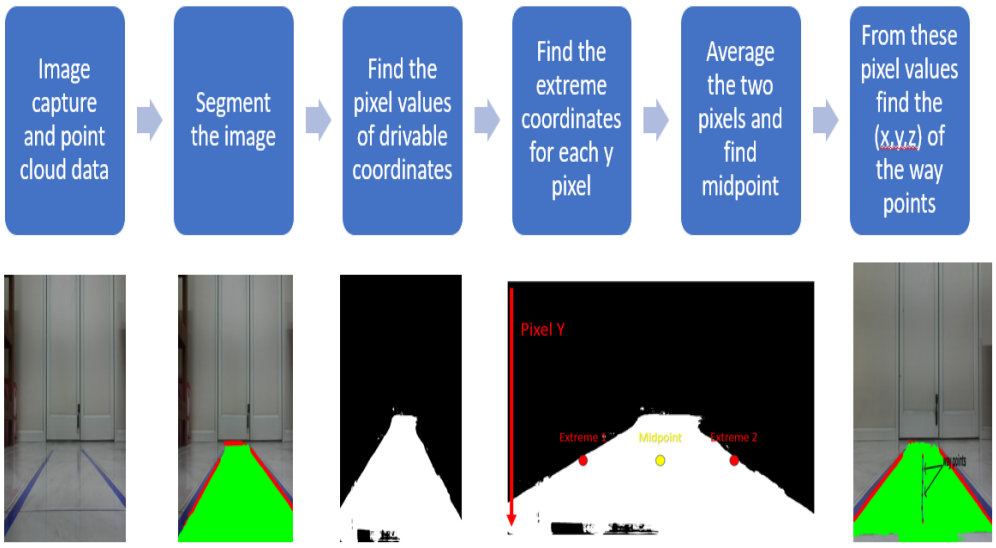


Figure 27: Flow chart

Error Control Mechanism

To ensure precise waypoint extraction, we've devised an error control mechanism. It addresses potential inaccuracies stemming from outliers or single noisy pixels by considering additional edge pixels within the drivable area. Our strategy employs a two-pointer approach for pinpointing extreme points amidst pixel groups.

Starting from the far left, our algorithm scrutinizes each pixel to ascertain its inclusion in the drivable area. Upon identifying a drivable pixel, we position both left and right pointers accordingly. These pointers then advance in unison towards the right until encountering five consecutive drivable pixels. Subsequently, the right pointer continues its progression independently as long as a continuous succession of drivable pixels persists.

This method guarantees a robust selection of extreme points, thus minimizing errors in the waypoint estimation process.

Algorithm 2 Algorithm for error control

Let $height$ and $width$ be the dimensions of the image
Create empty lists $left$, $right$, and $vertical$
for each y coordinate from the middle of the image to the bottom **do**
    Set $count$ and $flag$ to zero
    for each x coordinate from left to right **do**
        if the pixel at (x, y) is white (drivable) **then**
            Increment $count$ by 1
        else
            Reset $count$ to zero
        end if
        if $count$ reaches 5 and $flag$ is not set **then**
            Record x as the start of a drivable area
            Record x as the end of a drivable area
            Record y as the vertical position
            Set $flag$ to indicate drivable area found
        end if
        if $count$ exceeds 5 and $flag$ is set **then**
            Expand the recorded drivable area to include the current pixel
        end if
    end for
end for
Compute the midpoint $midx$ between the left and right boundaries for each drivable area
Convert the vertical positions $midy$ to integers

The efficacy of the two-pointer algorithm is demonstrated in Figure 30.

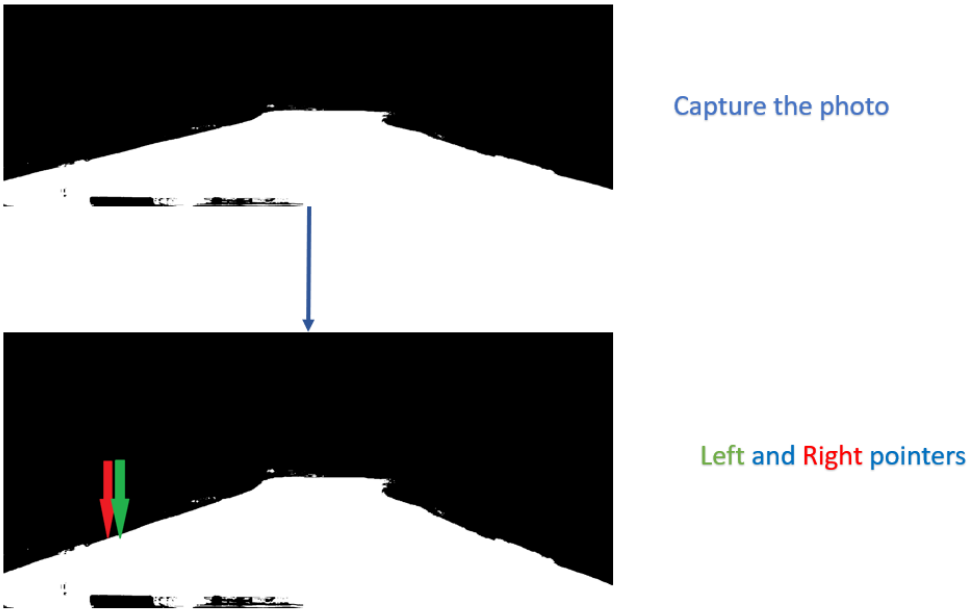


Figure 28: Capture the photo and fixing the left and right pointers

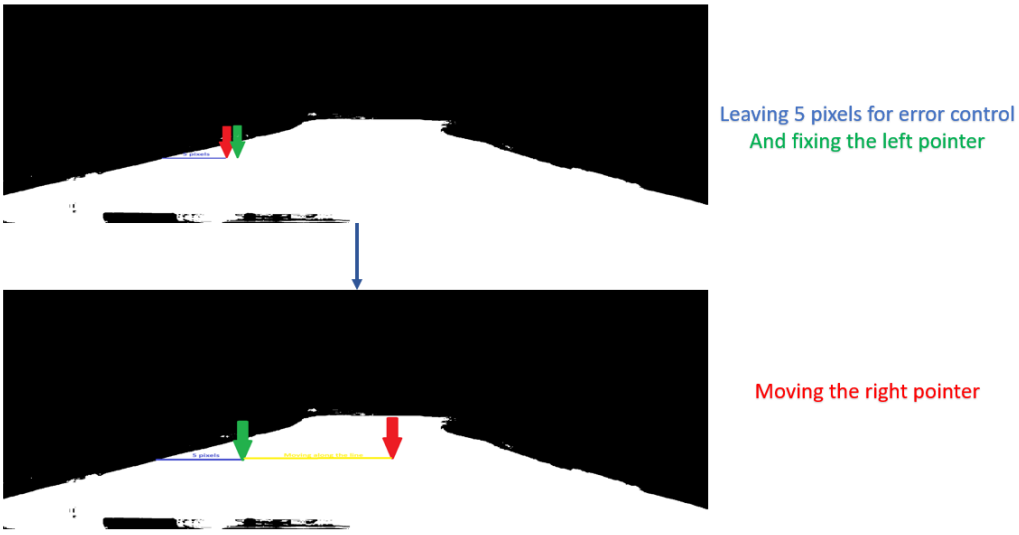


Figure 29: Leaving 5 pixels and moving the right pointer

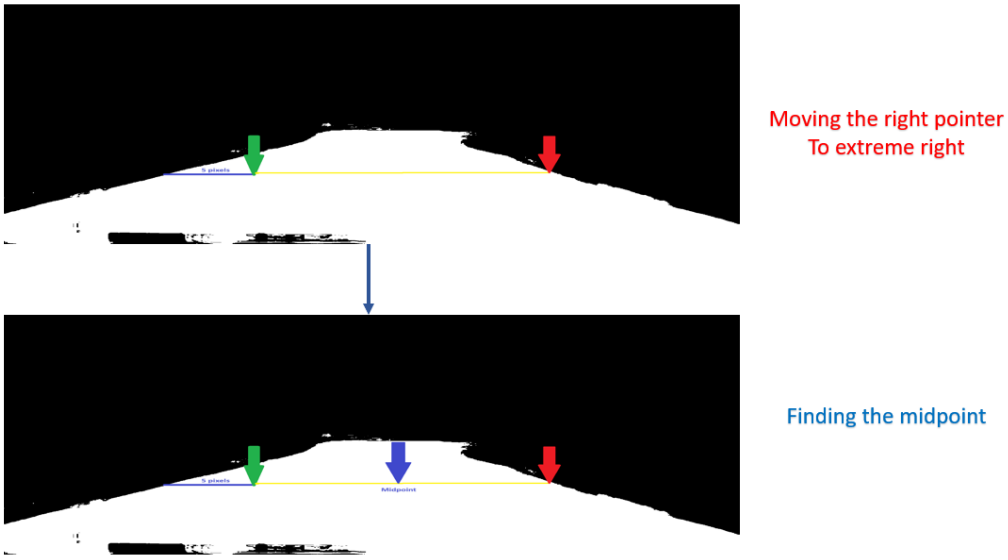


Figure 30: Moving the right pointer to extreme right and finding midpoint

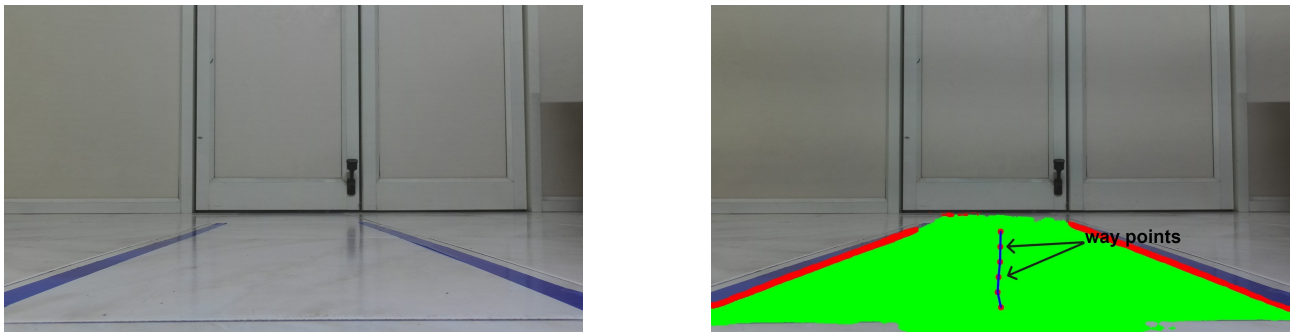
In this illustration, a horizontal line denotes the drivable area within an image. Initially positioned at the far left, two pointers—designated as left (L) and right (R)—commence their traversal.

Beginning with the left pointer, each pixel along the line is assessed for drivability. Upon detecting a drivable pixel, both pointers are anchored, proceeding jointly towards the right until encountering five consecutive drivable pixels. Once met, the right pointer advances independently, ensuring accurate identification of drivable points without succumbing to outlier influences.

Through the adoption of the two-pointer methodology, our algorithm navigates the drivable area adeptly, facilitating precise extraction of extreme points and enabling accurate waypoint estimation.

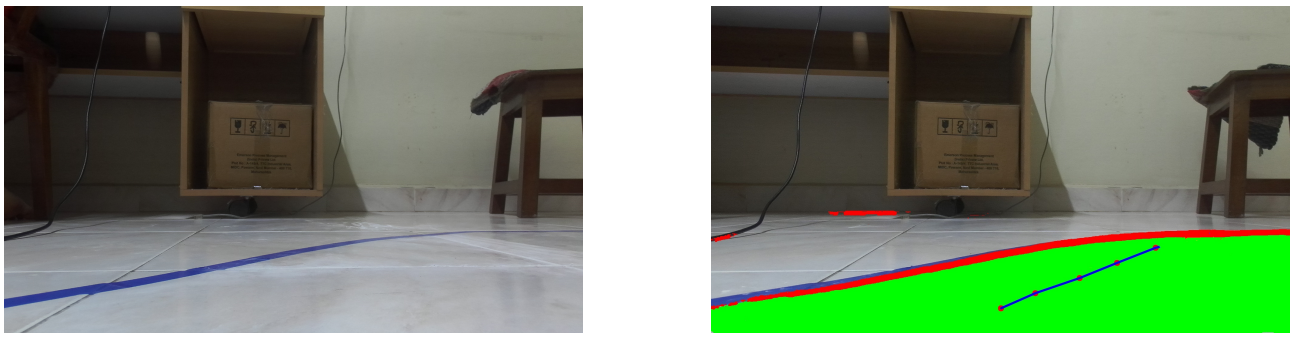
Results of the Path Planning Algorithm

In this section, we present the results of the path planning algorithm. The images below exhibit the original input images on the left side and the final processed images on the right side. The processing encompasses segmentation, path planning, and depth estimation. The green area represents the drivable area, while the red points denote the estimated waypoints. Additionally, the blue line overlaid on it indicates the path along which our car shall navigate.



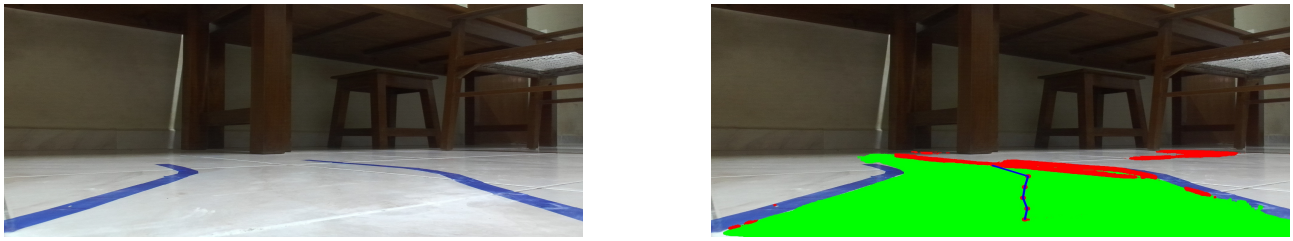
(a) Original Image (b) Processed Image

Figure 31: Straight Path



(a) Original Image (b) Processed Image

Figure 32: Right Turn



(a) Original Image (b) Processed Image

Figure 33: Left Turn

The images clearly demonstrate the effectiveness and accuracy of our algorithm. In the next section, we will present a probabilistic estimation of the algorithm's accuracy.

Probabilistic Estimation of the Accuracy of the Path Planning Algorithm

In this section, we present a probabilistic estimation of the accuracy of the path planning algorithm used in our system.

According to the YOLOP paper, the probability of a pixel being correctly labeled as drivable is 0.915.

$$P(\text{pixel correctly labeled as drivable}) = p_1 = 0.915$$

The probability of a pixel being incorrectly labeled as drivable is:

$$P(\text{pixel incorrectly labeled as drivable}) = p_2 = 1 - p_1 = 0.085$$

Since our algorithm requires 5 consecutive pixels to be incorrectly labeled in a continuous fashion to predict a wrong waypoint (assuming that prediction of each pixel as drivable or not is independent of each other), the probability of an incorrect waypoint is calculated as:

$$P(\text{Waypoint pixels incorrectly predicted}) = p_3 = (p_2)^5 = 4.44 \times 10^{-6}$$

Therefore, the probability of correctly predicting a waypoint is:

$$P(\text{Waypoint pixel correctly predicted}) = 1 - p_3 = 0.999995$$

Hence, the accuracy of the waypoint pixel calculation algorithm is 99.9995%.

Global Waypoint Generation

In this section, we introduce the concept of origin shifting to facilitate global waypoint generation. Origin shifting involves mapping all waypoints relative to a designated origin point, typically the starting position of the vehicle.

The algorithm employs origin shifting by estimating current waypoints relative to the vehicle's current position and then adding the coordinates of the current position calculated using the IMU sensor of the car. This approach ensures that the generated waypoints are referenced to a consistent global coordinate system, enabling accurate navigation.

The figure below illustrates the concept of origin shifting:

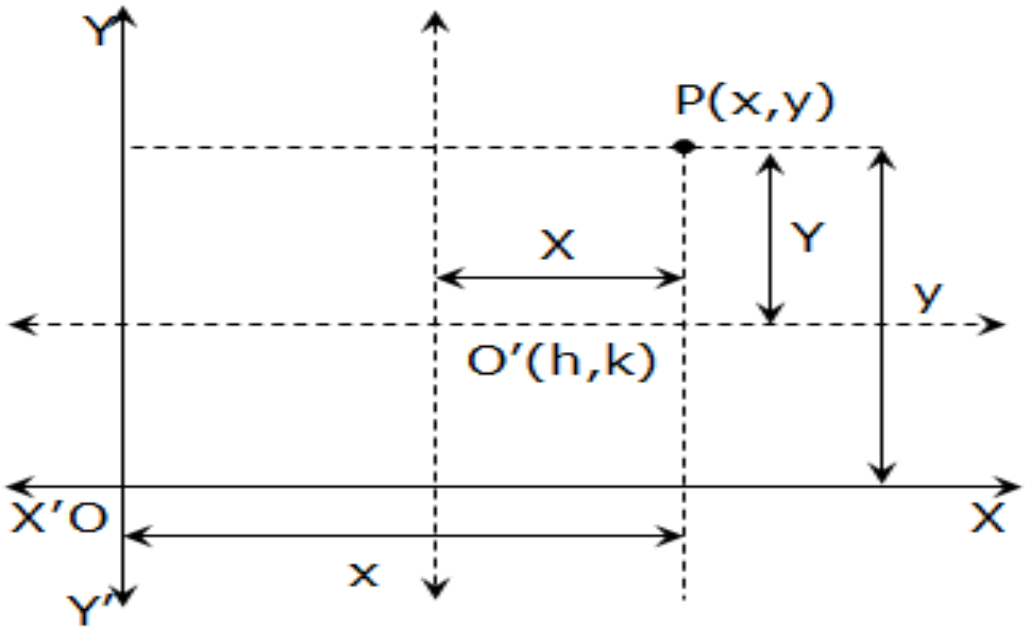


Figure 34: Illustration of Origin Shifting

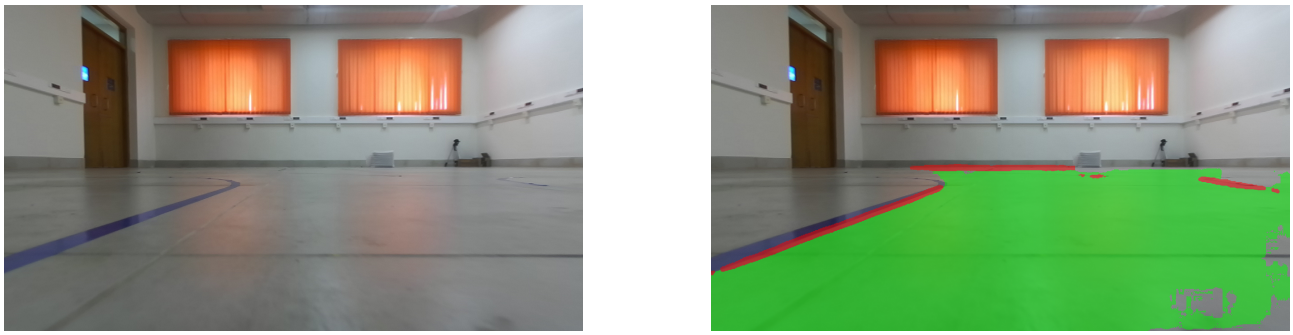
We developed the following algorithm using the concept of origin shifting for our car.

Algorithm 3 Global Waypoint Generation Algorithm

1: Define the origin point as the starting position of the vehicle.
2: Initialize empty lists to store relative waypoints and global waypoints.
3: Acquire current position coordinates using the IMU sensor.
4: **for** each detected waypoint **do**
5: Calculate the relative coordinates of the waypoint relative to the current position.
6: Add the current position to the relative waypoint coordinates to obtain global coordinates.
7: Append the global coordinates to the list of global waypoints.
8: **end for**
9: Output the list of global waypoints.

Summary of Results

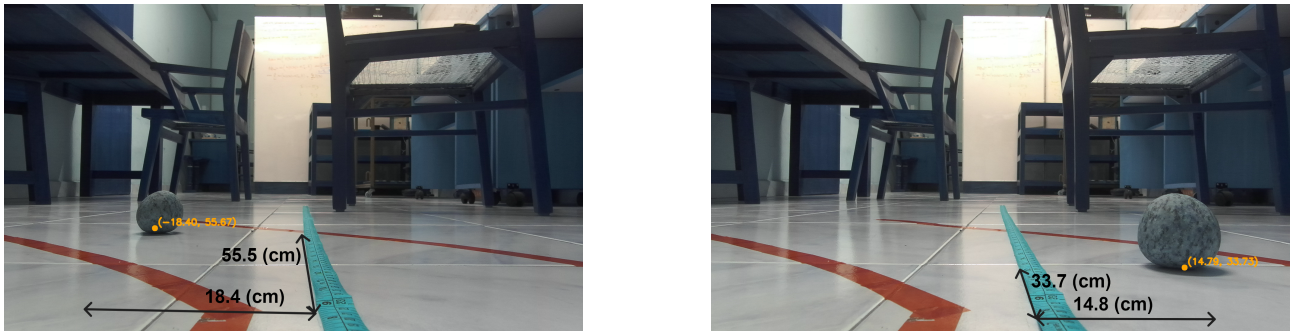
Here, we provide a concise summary of the results obtained throughout our experimentation process. Initially, we applied the YOLOP model for segmentation, integrating it into our car's system. The outcomes of this segmentation process are depicted below:



(a) Original Image (b) Segmented Output

Figure 35: Segmentation Results for a Left Turn

Subsequently, to address the depth estimation challenge, we employed inverse perspective mapping and ZED's point cloud method. The following figures showcase the outcomes of these methods:



(a) Depth Estimation Result (b) Ground Truth

Figure 36: Experiment Results using Inverse Perspective Mapping

Furthermore, the mean absolute error versus distance in z (cm) plots provide insights into the accuracy of depth estimation using inverse perspective mapping and ZED's point cloud method:

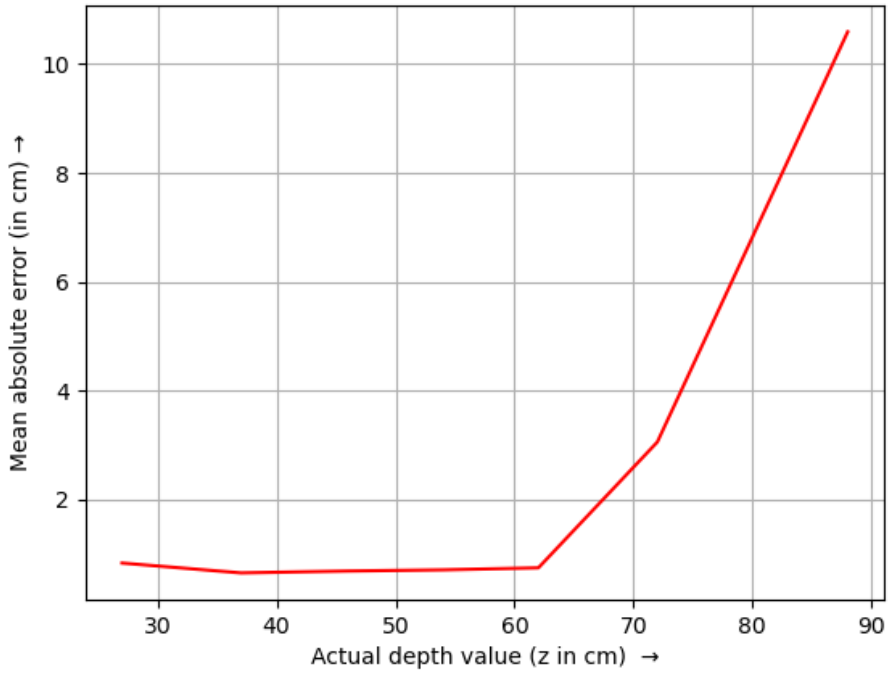


Figure 37: Mean Absolute Error vs. Distance using Inverse Perspective Mapping

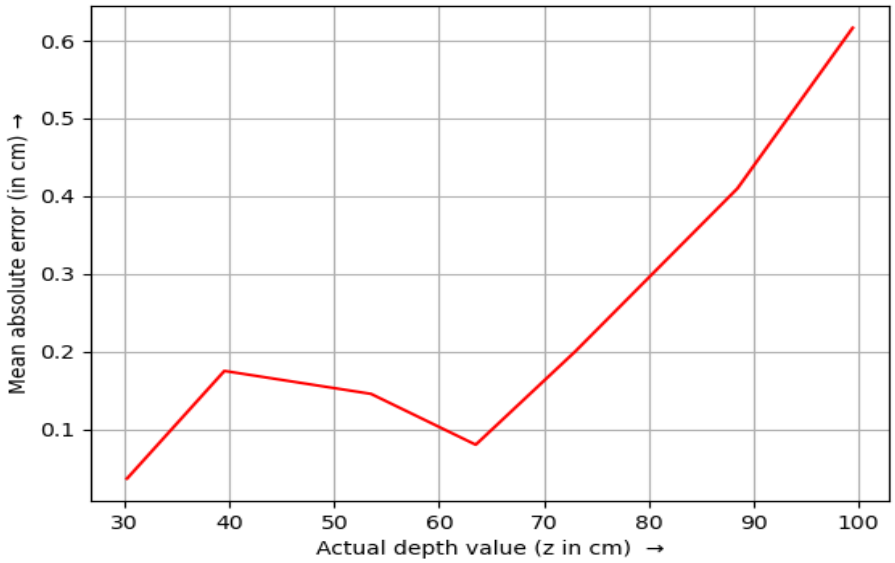
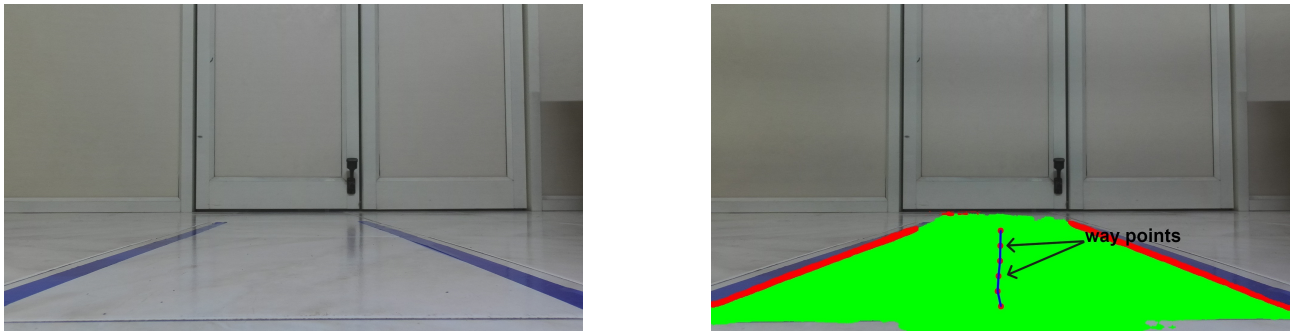


Figure 38: Mean Absolute Error vs. Distance using ZED's Point Cloud

Next, we developed a waypoint estimation algorithm incorporating the insights gained from previous steps. The outcomes of this algorithm are illustrated below:



(a) Original Image (b) Processed Image

Figure 39: Path Planning Results for a Straight Path

Finally, we evaluated the accuracy of the waypoint estimation algorithm, achieving an accuracy of 99.9995%.

This comprehensive analysis showcases the effectiveness of our approach in various aspects of autonomous navigation.

Additionally, the overall flow of the process is summarized in the flow chart below:

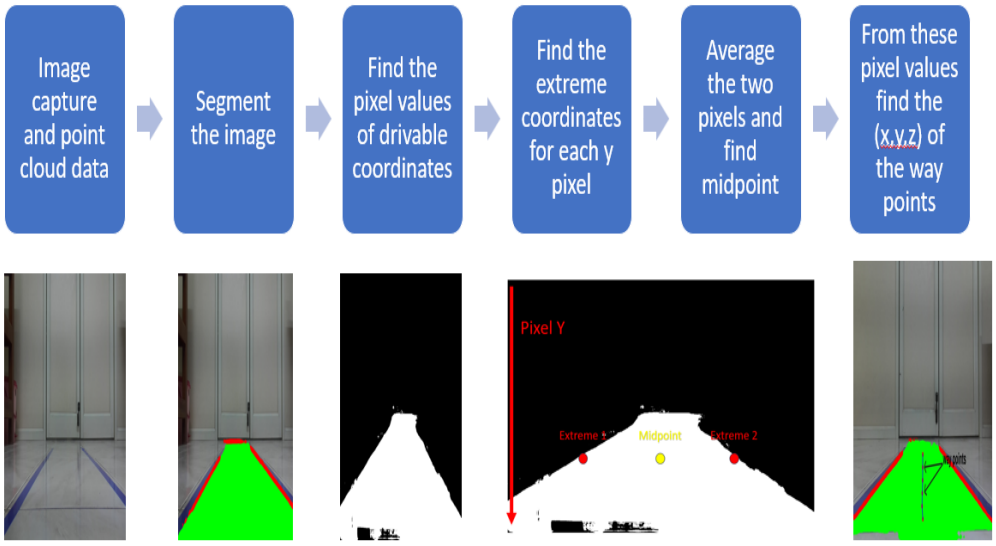


Figure 40: Flow chart

And after finding these way points releative to the current position we use the IMU sensors of the car to get the global way points.

Conclusion

In conclusion, this project successfully developed and integrated a sophisticated software module tailored for one-tenth scale autonomous vehicles. The utilization of the YOLOP model for scene segmentation enabled accurate identification of drivable areas, while the subsequent stages of waypoint generation and 3D coordinate estimation significantly enhanced the vehicle's navigation capabilities. Rigorous experimentation with various depth estimation methods ensured the reliability and accuracy of the navigation system.

The project demonstrated the efficacy of scaled-down vehicles in providing a controlled testing environment for autonomous driving algorithms. By leveraging the capabilities of V2V (vehicle-to-vehicle) and V2I (vehicle-to-infrastructure) communication, we were able to study cooperative autonomous vehicle behavior, traffic efficiency, and road safety within confined spaces. This controlled experimentation expedited algorithm validation and development, minimizing risks associated with full-scale testing.

Furthermore, this project highlighted the importance of interdisciplinary collaboration in driving innovation in autonomous vehicle technology. The integration of state-of-the-art algorithms and techniques, coupled with meticulous experimentation and validation, underscored our commitment to advancing knowledge in this domain. The insights gained from this research have broader implications for the fields of artificial intelligence and robotics, paving the way for more sophisticated and adaptive autonomous systems in the future.

Overall, this project represents a significant stride towards realizing fully autonomous transportation systems. The comprehensive software solution developed enhances the navigation prowess of autonomous vehicles, contributing to the continued evolution of autonomous vehicle technology and its broader applications in society.

Future work

Further, we plan to conduct an experiment on the car where we try to navigate the car over a track in the controlled environment of the lab. The path has already been constructed, and with the help of the control module, we will be able to navigate the car and validate the effectiveness of these algorithms. Additionally, we will provide clear documentation of the code to ensure it becomes easy for future developers to use and modify the code.

To enhance the functionality, our developed algorithms can seamlessly integrate with the car's control module for autonomous navigation. We provide comprehensive documentation on how to utilize the code and ensure that it is thoroughly commented, making it easily understandable for control engineers. This facilitates seamless integration of the algorithms into the car's system for autonomous driving. The image below shows the track and the car.

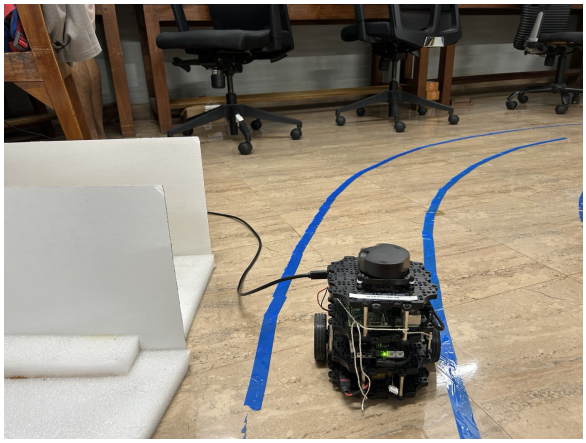


Figure 41: Track and the Turtle bot

For detailed instructions and access to the code, please refer to the provided link in the footnote: [8]

References

1. Dong Wu, Manwen Liao, Weitian Zhang, Xinggang Wang, Xiang Bai, Wenqing Cheng, Wenyu Liu, *YOLOP: You Only Look Once for Panoptic Driving Perception*, 2022. link (paper)

2. Wouter Van Gansbeke, Bert De Brabandere, Davy Neven, Marc Proesmans, and Luc Van Gool, *End-to-end lane detection through differentiable least-squares fitting. 2019.*, 2022. link (paper)

[8] Link to code: here