Obstacle Avoidance and Navigation

Problem statement

The problem is to develop an obstacle avoidance system for a vehicle that can effectively detect and navigate around obstacles in its path, ensuring safe and efficient movement.

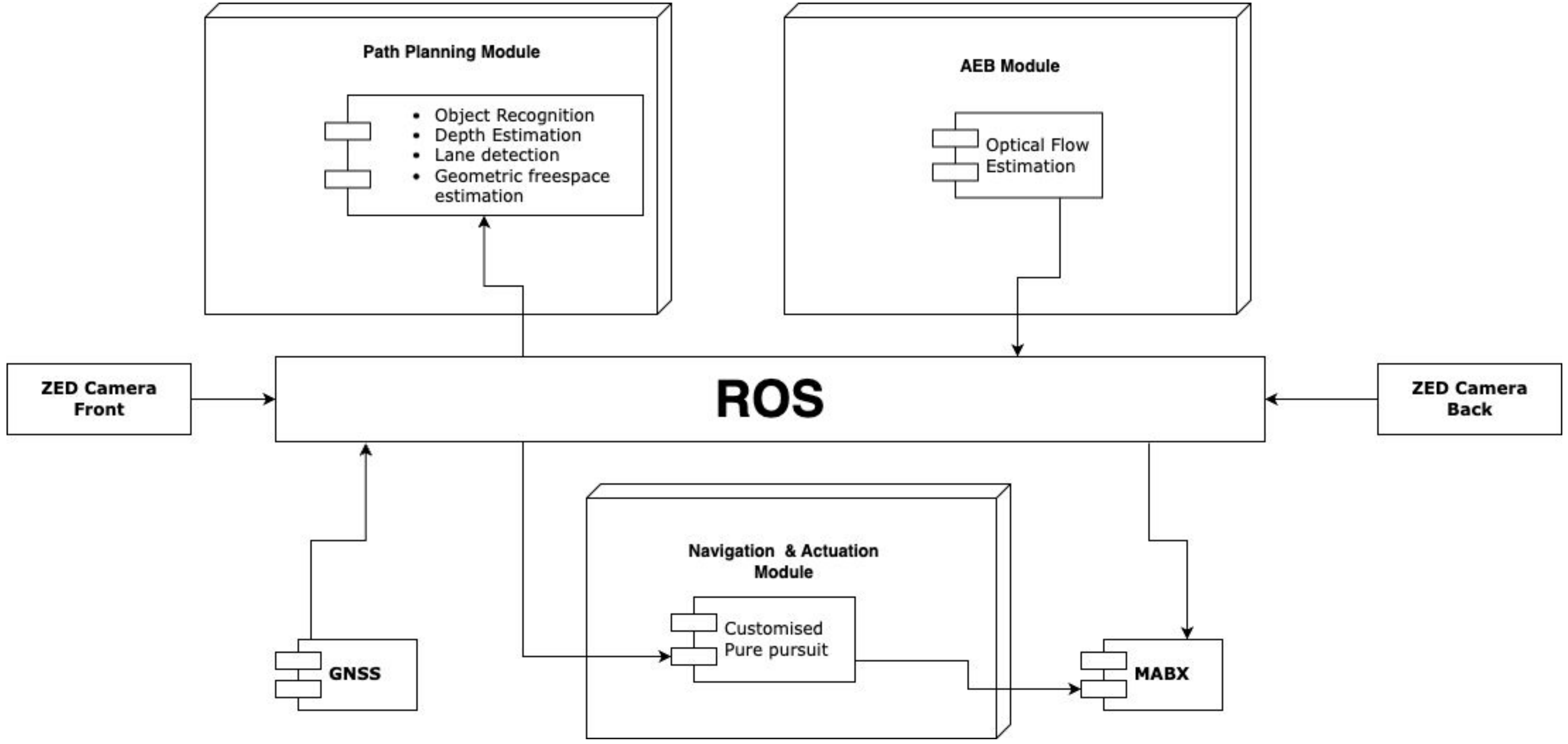
The system should be able to handle various scenarios and challenges, including dynamic and static obstacles, and provide real-time decision-making capabilities to avoid collisions and ensure smooth navigation.

TiHAN-IIT Hyderabad

NextGen Autonomous Navigation

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Proposed High Level System Design



SUZUKI

Testing scenarios

Scenario 1: Vehicle Approaching from Behind:

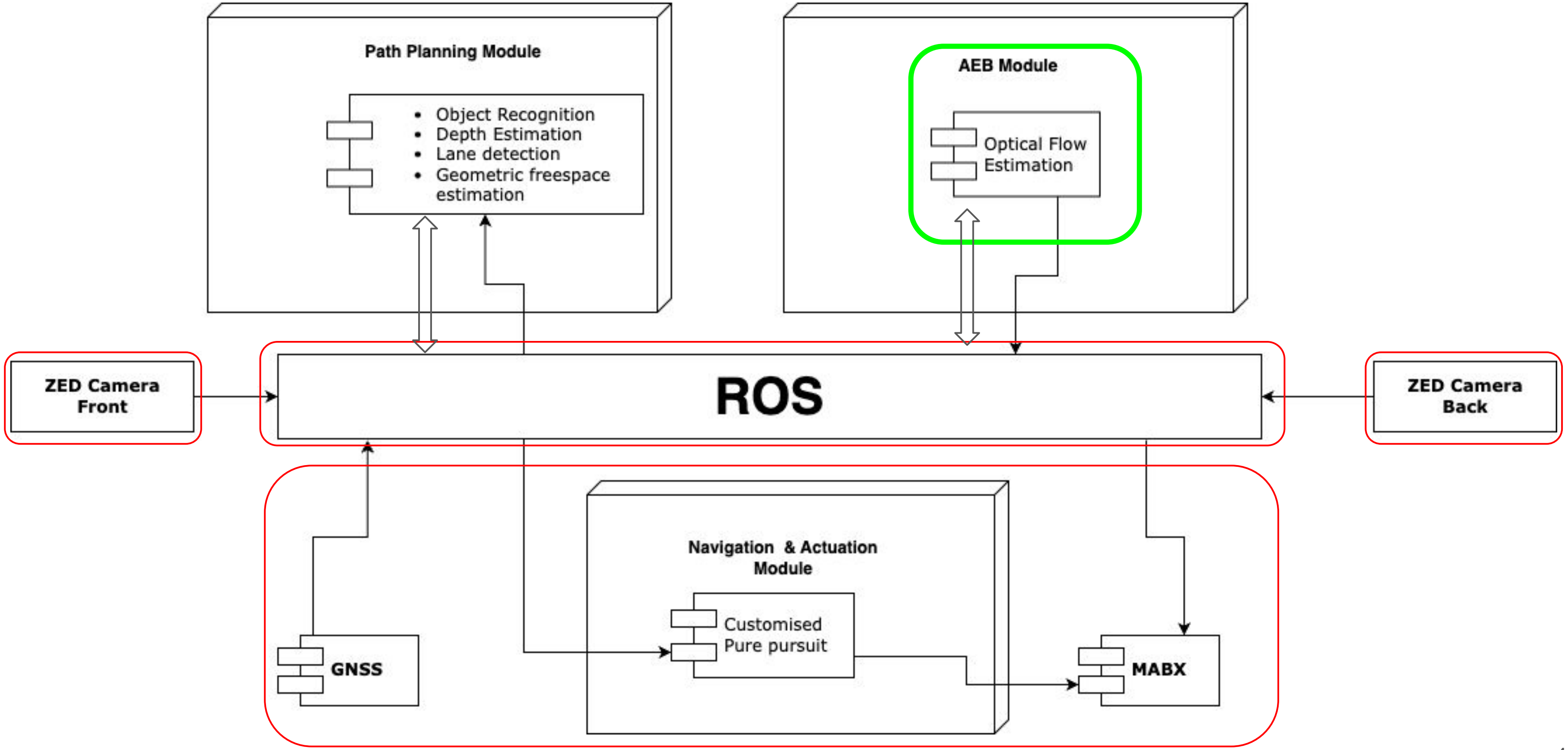
In this scenario, the vehicle equipped with the obstacle avoidance system is driving on a road, and another vehicle is approaching from behind at a higher speed. The system needs to detect the approaching vehicle and assess the risk of a potential collision. It should generate appropriate navigation commands, such as steering or acceleration adjustments, to avoid the approaching vehicle while maintaining a safe distance and smooth trajectory.

Scenario 2: Pedestrian Crossing:

In this scenario, the vehicle encounters a pedestrian crossing the road from a side street. The system should be able to detect the pedestrian and accurately estimate their position and motion. It needs to make timely decisions to avoid colliding with the pedestrian, potentially by slowing down or altering the vehicle's trajectory. The system should ensure that the pedestrian has safely crossed before resuming its intended path.

High Level System Design - update

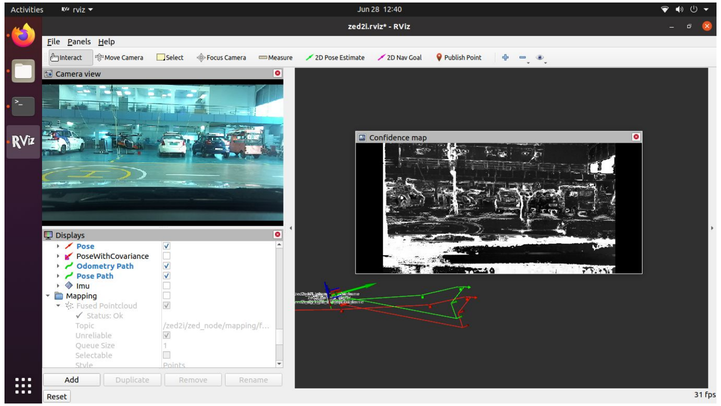
TiHAN-IIT Hyderabad
NextGen Autonomous Navigation
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad



SUZUKI

ZED camera integration with ROS - Depth

TiHAN-IIT Hyderabad
NextGen Autonomous Navigation
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad



Displays	
Pose	✓
PoseWithCovariance	☐
Odometry Path	✓
Pose Path	✓
Imu	☐
Mapping	☐
Fused Pointcloud	✓
✓ Status: Ok	
Topic	/zed2i/zed_node/mapping/f...
Unreliable	✓
Queue Size	1
Selectable	☐
Style	Points

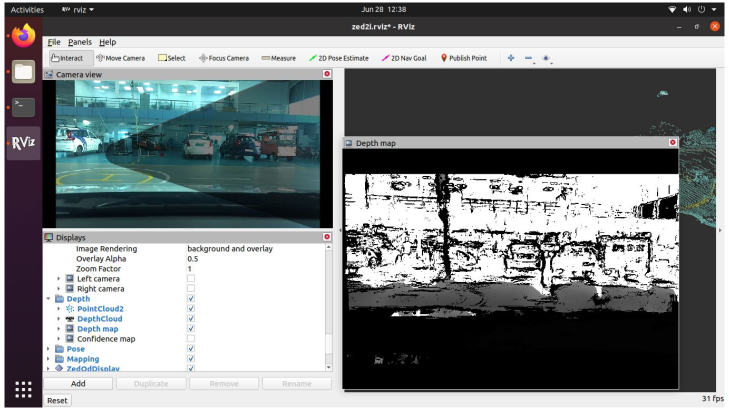
Depth Range

Depth range corresponds to the minimum and maximum distance at which the depth of an object can be estimated.

DEPTH RANGE	ZED 2	ZED	ZED MINI
Default	0.4m to 20m (1.3 to 65ft)	0.4m to 25m (1.3 to 82ft)	0.2m to 15m (0.6 to 49ft)
Min range	0.3m (1.0ft)	0.2m (0.6ft)	0.1m (0.3ft)
Max range	40m (131ft)	40m (131ft)	20m (65ft)

Confidence map for depth visualized via rviz:
confidence map gives every pixel (X, Y) in the image a value in the range [1,100].

ZED stereo camera with ROS with ZED SDK 4.0 - outputs the camera left and right images, depth map, point cloud, pose information and supports the use of multiple ZED cameras.



Displays	
Image Rendering	background and overlay
Overlay Alpha	0.5
Zoom Factor	1
Left camera	☐
Right camera	☐
Depth	✓
PointCloud2	✓
DepthCloud	✓
Depth map	✓
Confidence map	☐
Pose	✓
Mapping	✓
ZedOdDisplay	✓

Motion Estimation using Optical Flow

- Motion estimation: Process of determining the apparent motion of objects between consecutive frames in a video.
- Optical flow: Technique for estimating motion by analyzing the pixel intensity patterns between frames.

The estimated optical flow provides valuable information about the speed, direction, and trajectory of objects in the video.

Optical Flow Computation

Assumptions:

- Brightness constancy: Pixel intensity remains constant across frames.
- Spatial coherence: Nearby pixels have similar motion.
- Temporal consistency: Motion is consistent over time.

Optical Flow Equation:

$$\frac{\partial I}{\partial x} \cdot v_x + \frac{\partial I}{\partial y} \cdot v_y + \frac{\partial I}{\partial t} = 0$$

Where,

$\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}, and \frac{\partial I}{\partial t}$ Image gradients in the x, y, and time dimensions.

$v_x \; and \; v_y$ Optical flow velocities in the x and y directions.

Optical Flow Computation Techniques

TiHAN-IIT Hyderabad

NextGen Autonomous Navigation

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Dense Optical Flow:

- Estimates motion vectors for every pixel in the image.
  - Output: The result is a dense flow field where each pixel has an associated motion vector.
  - Computation: Dense optical flow requires calculations for all pixels, making it computationally more expensive.
- Methods: Lucas-Kanade, Horn-Schunck, Farneback, etc.

Sparse Optical Flow:

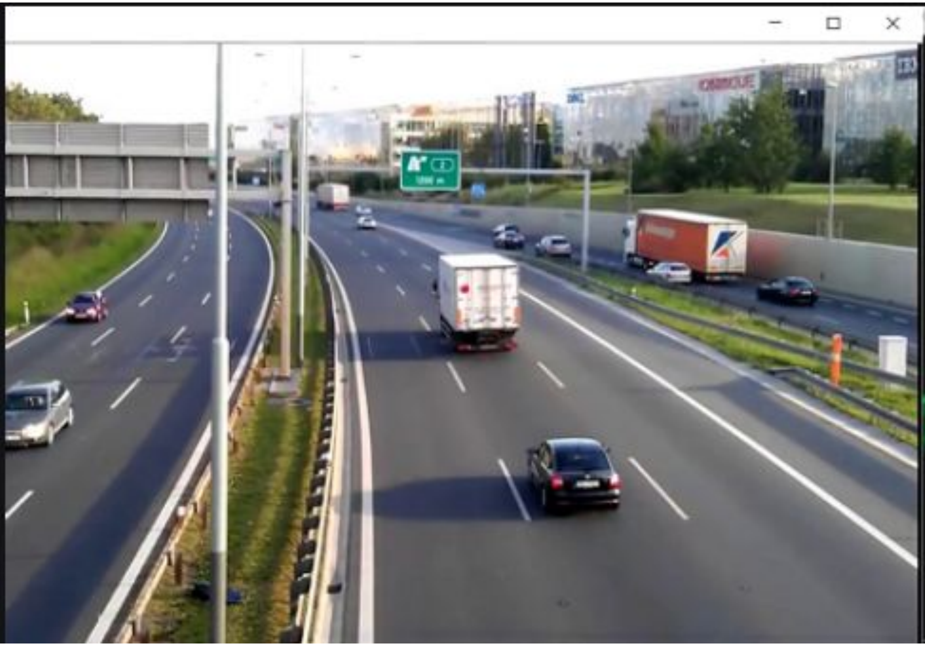
- Estimates motion only for a selected set of feature points.
  - Calculation: It estimates the motion only at specific points of interest, typically determined by feature detection algorithms.
  - Output: The result is a sparse set of motion vectors associated with the detected feature points.
- Methods: Kanade-Lucas-Tomasi (KLT), Shi-Tomasi, Good Features to Track (GFTT), etc.

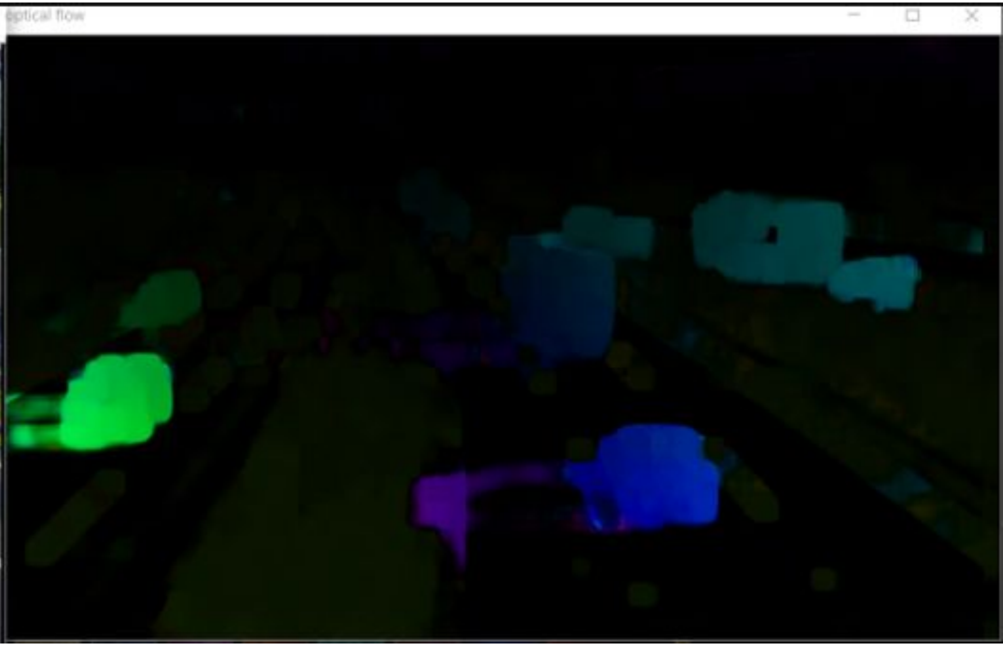
TiHAN-IIT Hyderabad

NextGen Autonomous Navigation

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Sample output - Dense





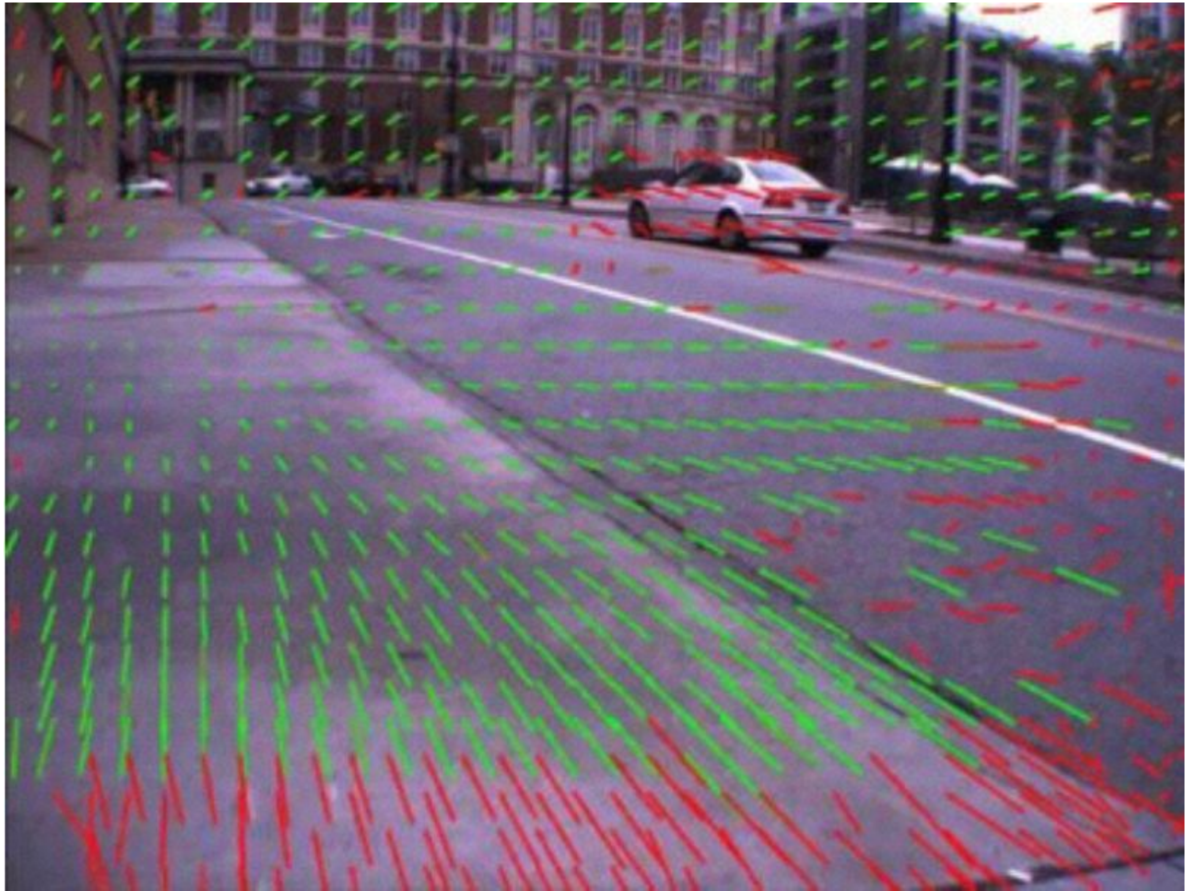
SUZUKI

Sample output - Sparse

TiHAN-IIT Hyderabad

NextGen Autonomous Navigation

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

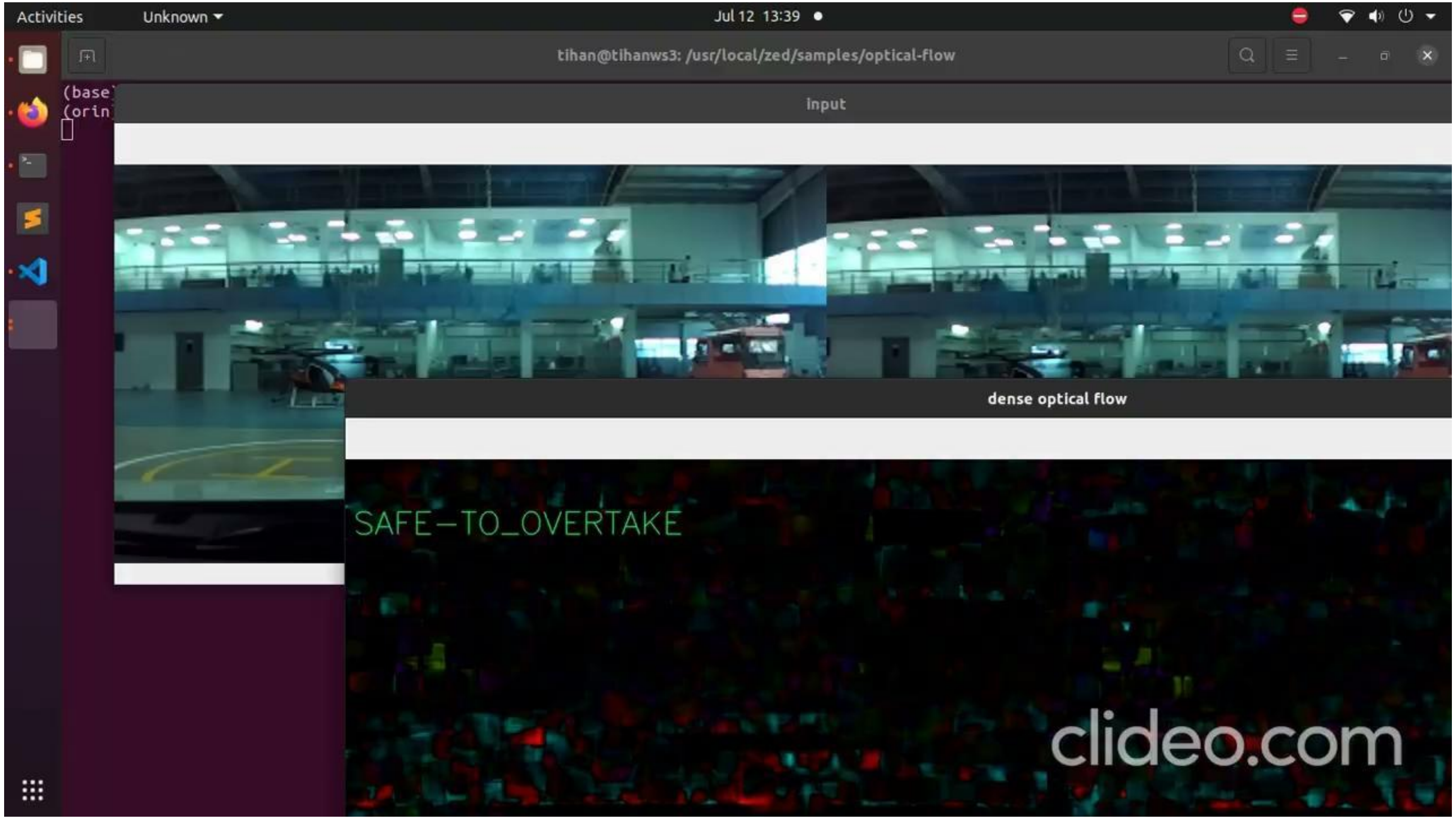


Optical Flow - Obstacle movement prediction

TiHAN-IIT Hyderabad

NextGen Autonomous Navigation

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad



```
(base)
(orin
```

```
SAFE—TO_OVERTAKE
```

Link: https://drive.google.com/file/d/1FPvYxXkDWW_gjvPbj83Z8yZar-9_0cQX/view?usp=sharing

Algorithm workflow

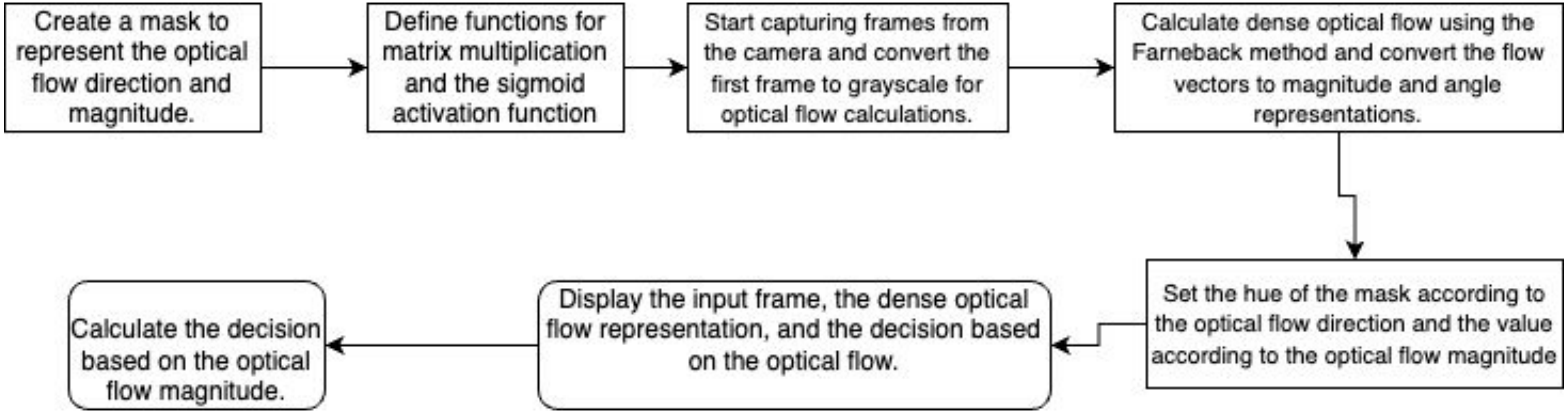

Create a mask to represent the optical flow direction and magnitude.
Define functions for matrix multiplication and the sigmoid activation function
Start capturing frames from the camera and convert the first frame to grayscale for optical flow calculations.
Calculate dense optical flow using the Farneback method and convert the flow vectors to magnitude and angle representations.
Set the hue of the mask according to the optical flow direction and the value according to the optical flow magnitude
Display the input frame, the dense optical flow representation, and the decision based on the optical flow.
Calculate the decision based on the optical flow magnitude.

Fig.: Obstacle movement tracking using Optical Flow in steps

TiHAN-IIT Hyderabad
NextGen Autonomous Navigation

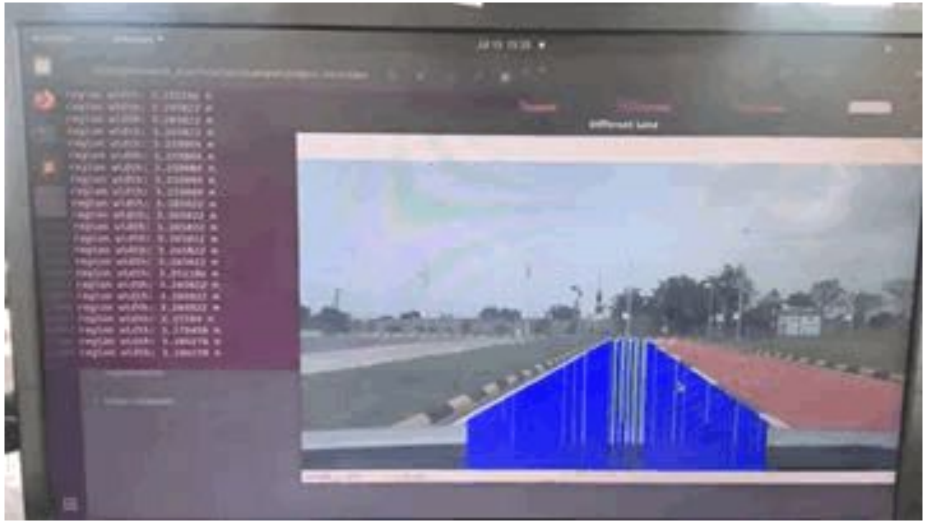
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

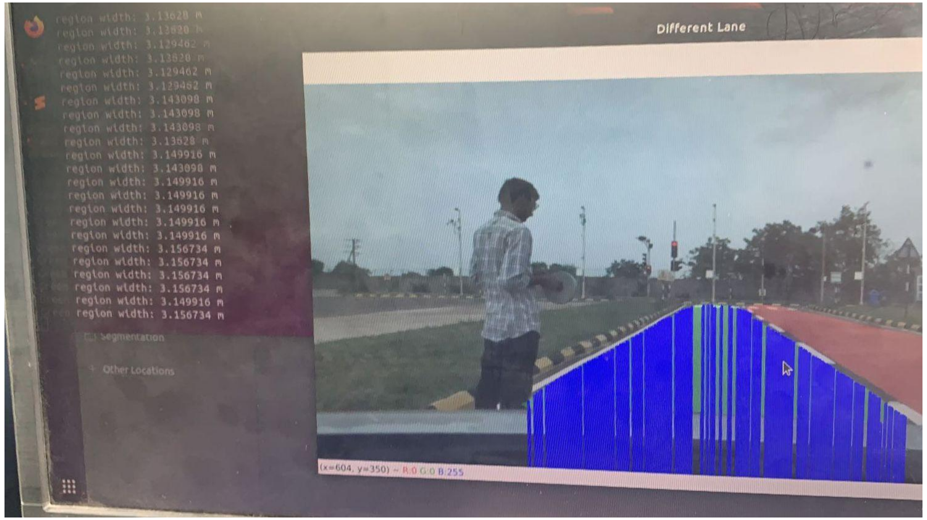
Algorithm features

The algorithm

- captures frames from the camera in real-time,
- calculates dense optical flow, and
- determines the direction of the optical flow based on the magnitude.
- displays the optical flow and decision information on the video feed.
- provides real-time analysis of optical flow and aids in decision-making based on the flow direction

Segmentation based Lane Width Result



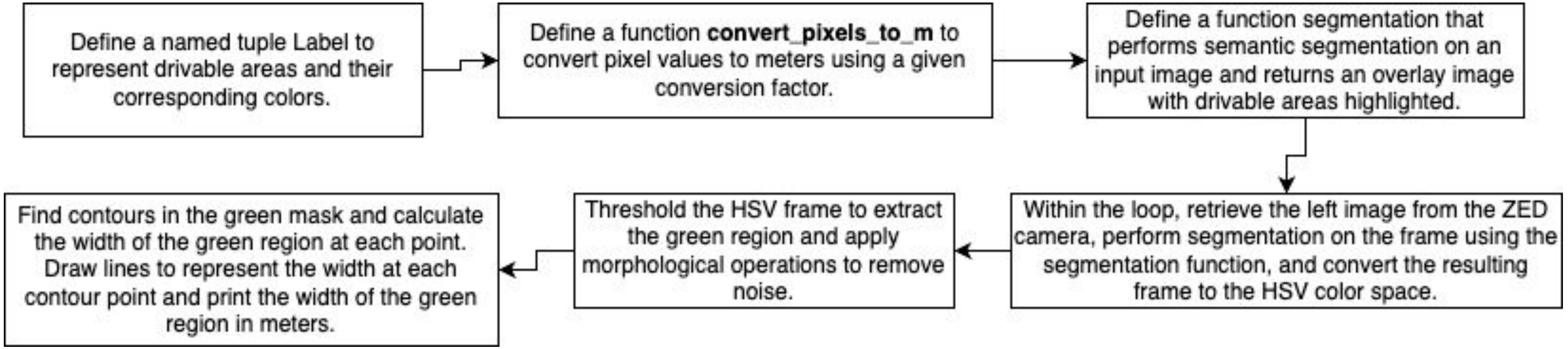


```
region width: 3.13628 m
region width: 3.13628 m
region width: 3.129462 m
region width: 3.13628 m
region width: 3.129462 m
region width: 3.129462 m
region width: 3.143098 m
region width: 3.143098 m
region width: 3.143098 m
region width: 3.13628 m
region width: 3.149916 m
region width: 3.143098 m
region width: 3.149916 m
region width: 3.149916 m
region width: 3.149916 m
region width: 3.149916 m
region width: 3.149916 m
region width: 3.156734 m
region width: 3.156734 m
region width: 3.156734 m
region width: 3.156734 m
region width: 3.149916 m
region width: 3.156734 m
```

Fig. Accurately calculating Lane width

SUZUKI

Segmentation based Lane Width Algorithm



Define a named tuple Label to represent drivable areas and their corresponding colors.

Define a function **convert_pixels_to_m** to convert pixel values to meters using a given conversion factor.

Define a function segmentation that performs semantic segmentation on an input image and returns an overlay image with drivable areas highlighted.

Within the loop, retrieve the left image from the ZED camera, perform segmentation on the frame using the segmentation function, and convert the resulting frame to the HSV color space.

Threshold the HSV frame to extract the green region and apply morphological operations to remove noise.

Find contours in the green mask and calculate the width of the green region at each point. Draw lines to represent the width at each contour point and print the width of the green region in meters.

Algorithm summarization

TiHAN-IIT Hyderabad

NextGen Autonomous Navigation

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

The code

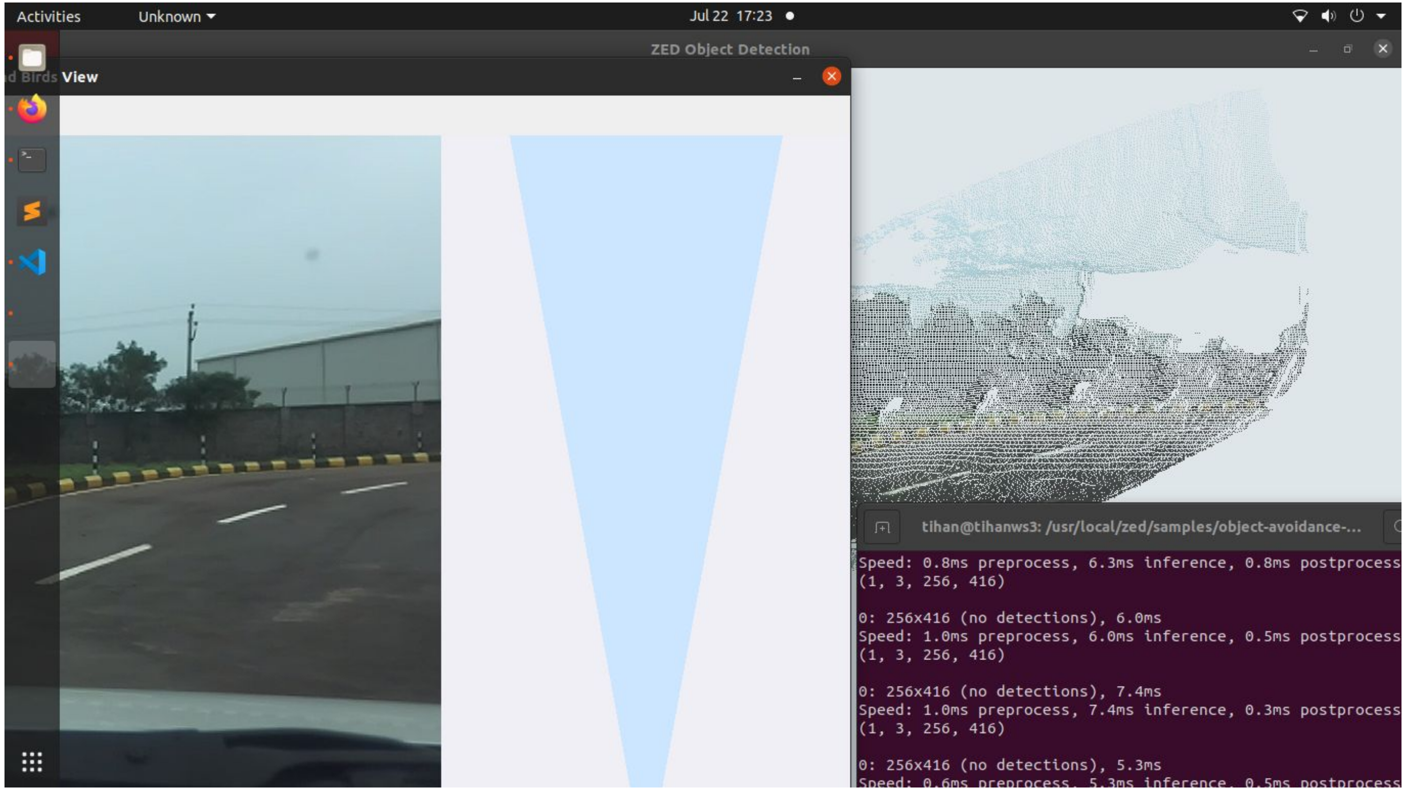
- uses the PyZED SDK to interact with a ZED camera,
- performs semantic segmentation using a PSPNet model to detect drivable regions, and
- calculates the width of the green region in meters at each point on the drivable lanes.
- The resulting frame with highlighted green lanes is displayed in real-time

3D point cloud & 2D viz - ZED + YOLOv8

TiHAN-IIT Hyderabad

NextGen Autonomous Navigation

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad



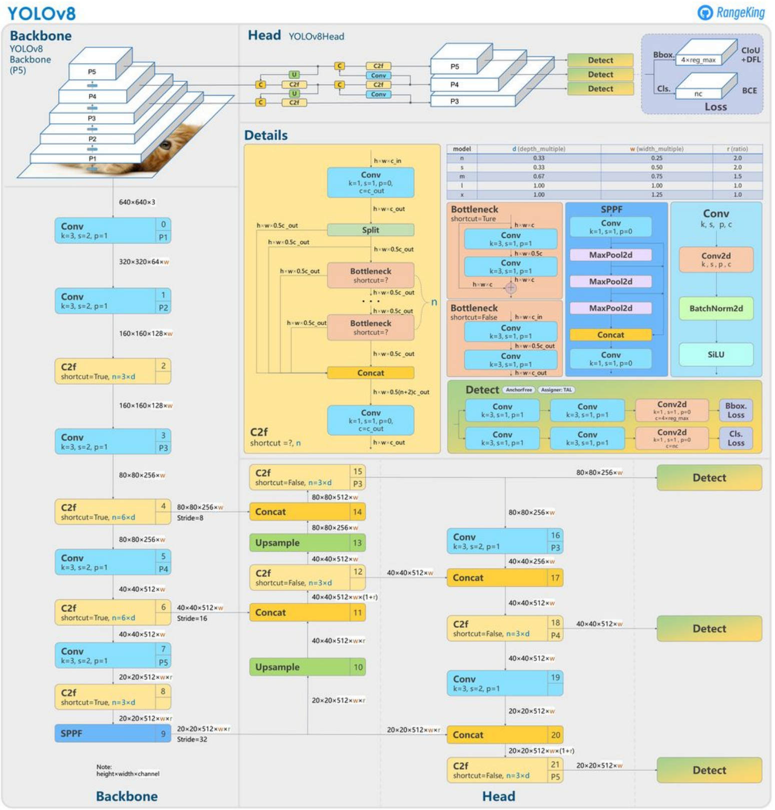
```
Speed: 0.8ms preprocess, 6.3ms inference, 0.8ms postprocess
(1, 3, 256, 416)

0: 256x416 (no detections), 6.0ms
Speed: 1.0ms preprocess, 6.0ms inference, 0.5ms postprocess
(1, 3, 256, 416)

0: 256x416 (no detections), 7.4ms
Speed: 1.0ms preprocess, 7.4ms inference, 0.3ms postprocess
(1, 3, 256, 416)

0: 256x416 (no detections), 5.3ms
Speed: 0.6ms preprocess, 5.3ms inference, 0.5ms postprocess
```

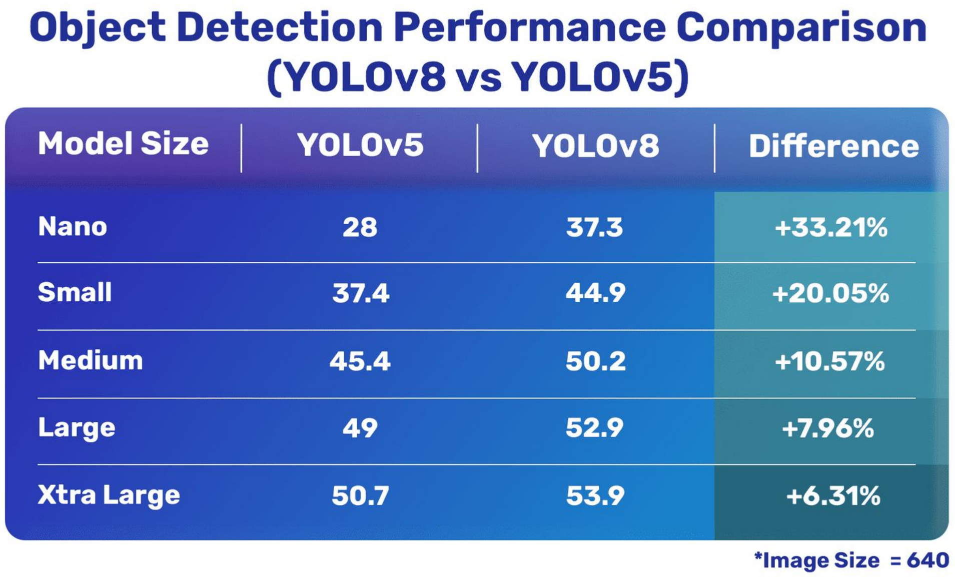
Fig. Showing 3D point cloud, 2D cv2 and BEV with detections in terminal

SUZUKI

YOLOv8 model with ZED - Depth, Point Cloud

TiHAN-IIT Hyderabad
NextGen Autonomous Navigation
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad





Object Detection Performance Comparison (YOLOv8 vs YOLOv5)

Model Size	YOLOv5	YOLOv8	Difference
Nano	28	37.3	+33.21%
Small	37.4	44.9	+20.05%
Medium	45.4	50.2	+10.57%
Large	49	52.9	+7.96%
Xtra Large	50.7	53.9	+6.31%

*Image Size = 640

Ref: https://blog.roboflow.com/whats-new-in-yolov8/ , https://docs.ultralytics.com/ , https://www.augmentedstartups.com/blog/yolov8-vs-yolov5-choosing-the-best-object-detection-model

SUZUKI

Algorithm overview

TiHAN-IIT Hyderabad

NextGen Autonomous Navigation

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Importing Libraries: The code imports the necessary libraries, including PyZED SDK (for interfacing with the ZED camera), Ultralytics (for YOLO object detection), OpenCV (for image processing and visualization), and PyTorch (for deep learning).

torch_thread: This function runs on a separate thread and performs object detection using the YOLO model loaded from the specified weights file. It continuously updates the detections variable with the detected objects' information.

main: The main function initializes the ZED camera, object detection parameters, and the OpenGL viewer. It also starts the torch_thread in a separate thread. The main loop grabs images from the camera, runs object detection, retrieves objects' data, and updates the OpenGL and OpenCV visualizations with the detected objects' bounding boxes.

Display and Visualization: After the torch_thread completes the detection, the main thread retrieves the detected objects' information from the detections variable. It updates the 3D point cloud visualization and 2D image display with the detected objects' bounding boxes.

Helper Functions:

- **xywh2abcd**: Converts bounding box coordinates from (x, y, width, height) to (A, B, C, D) format, representing four corners of the bounding box.
- **get_class_label**: Extracts the class label (object category) from the detections.
- **detections_to_custom_box**: Converts YOLO detections to ZED SDK custom box format for object tracking.

- Initializing ZED Camera and OpenGL Viewer: The function starts by initializing the ZED camera. It sets up the camera parameters, depth mode, coordinate system, and other required settings for capturing images and 3D point clouds. The ZED camera is opened using the sl.Camera object.

- Enabling Positional Tracking and Object Detection: The function enables positional tracking using sl.enable_positional_tracking() and sets up object detection parameters using sl.enable_object_detection().

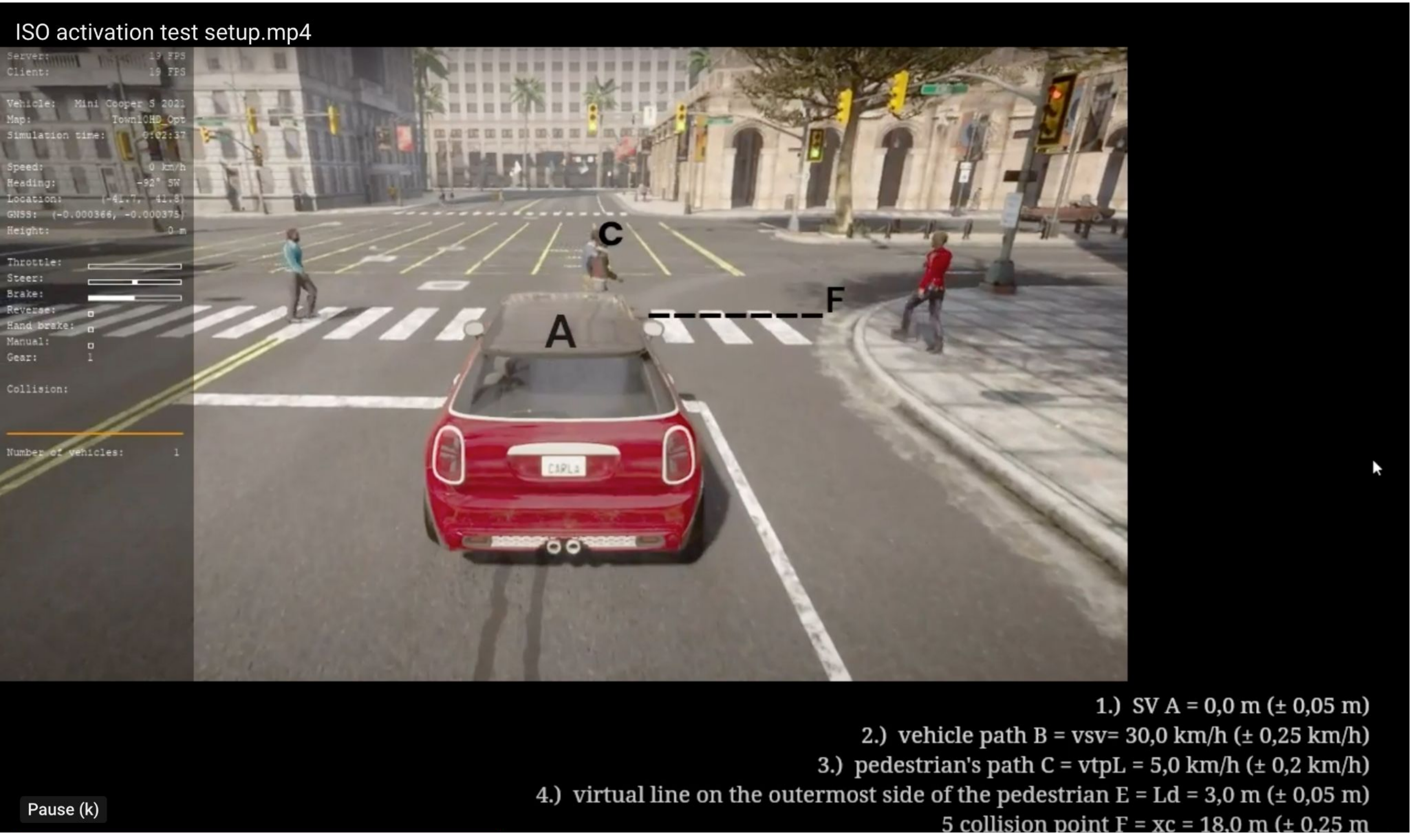
- Visualizations and Rendering: The function initializes OpenGL and OpenCV viewers to visualize the camera's 3D point cloud and 2D images with detected objects' bounding boxes. It sets up parameters for 2D display and camera tracking view.

- Main Loop: The main loop starts, and the program enters real-time object detection mode. It repeatedly grabs images from the ZED camera using zed.grab() and runs the object detection process.

TiHAN-IIT Hyderabad

NextGen Autonomous Navigation

CARLA Simulations


ISO activation test setup.mp4

```
Server:                19 FPS
Client:                19 FPS

Vehicle:    Mini Cooper S 2021
Map:              Town10HD_Opt
Simulation time:       0:02:37

Speed:                  0 km/h
Heading:               -92° SW
Location:        (-41.7,  41.8)
GNSS:  (-0.000366, -0.000375)
Height:                    0 m

Throttle:
Steer:
Brake:
Reverse:      □
Hand brake:   □
Manual:       □
Gear:         1

Collision:

Number of vehicles:          1
```

1.) SV A = 0,0 m (± 0,05 m)
2.) vehicle path B = vsv= 30,0 km/h (± 0,25 km/h)
3.) pedestrian's path C = vtpL = 5,0 km/h (± 0,2 km/h)
4.) virtual line on the outermost side of the pedestrian E = Ld = 3,0 m (± 0,05 m)
5 collision point F = xc = 18,0 m (± 0,25 m

Pause (k)

Link: https://drive.google.com/file/d/14-Vu1Ym-eV4PpqwkpWv7kVdT4CcxeBn-/view?usp=sharing

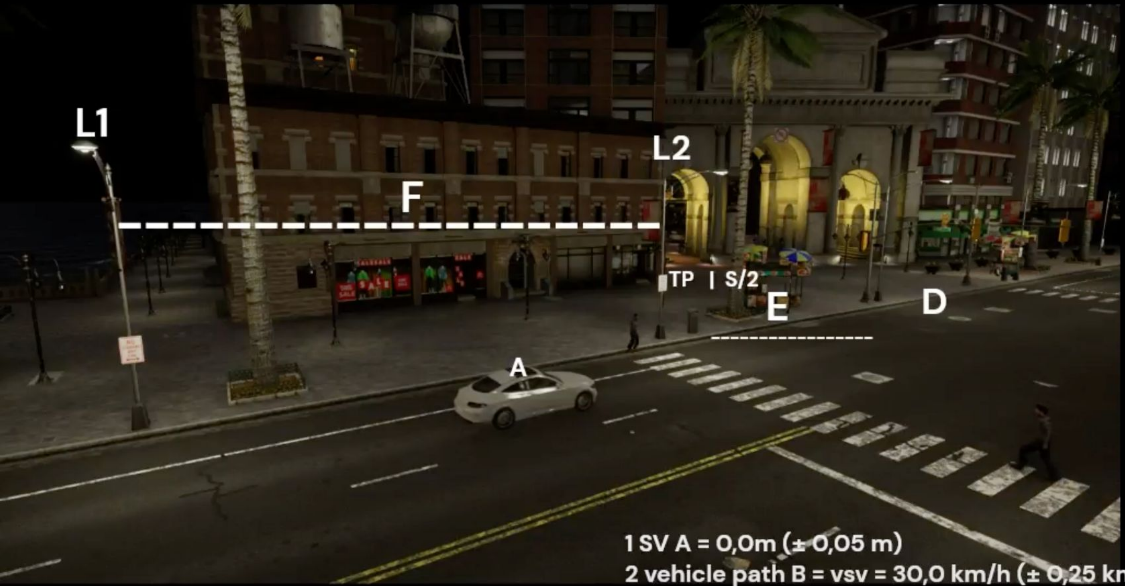
TiHAN-IIT Hyderabad

NextGen Autonomous Navigation

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

CARLA Simulations

ISO II.mp4

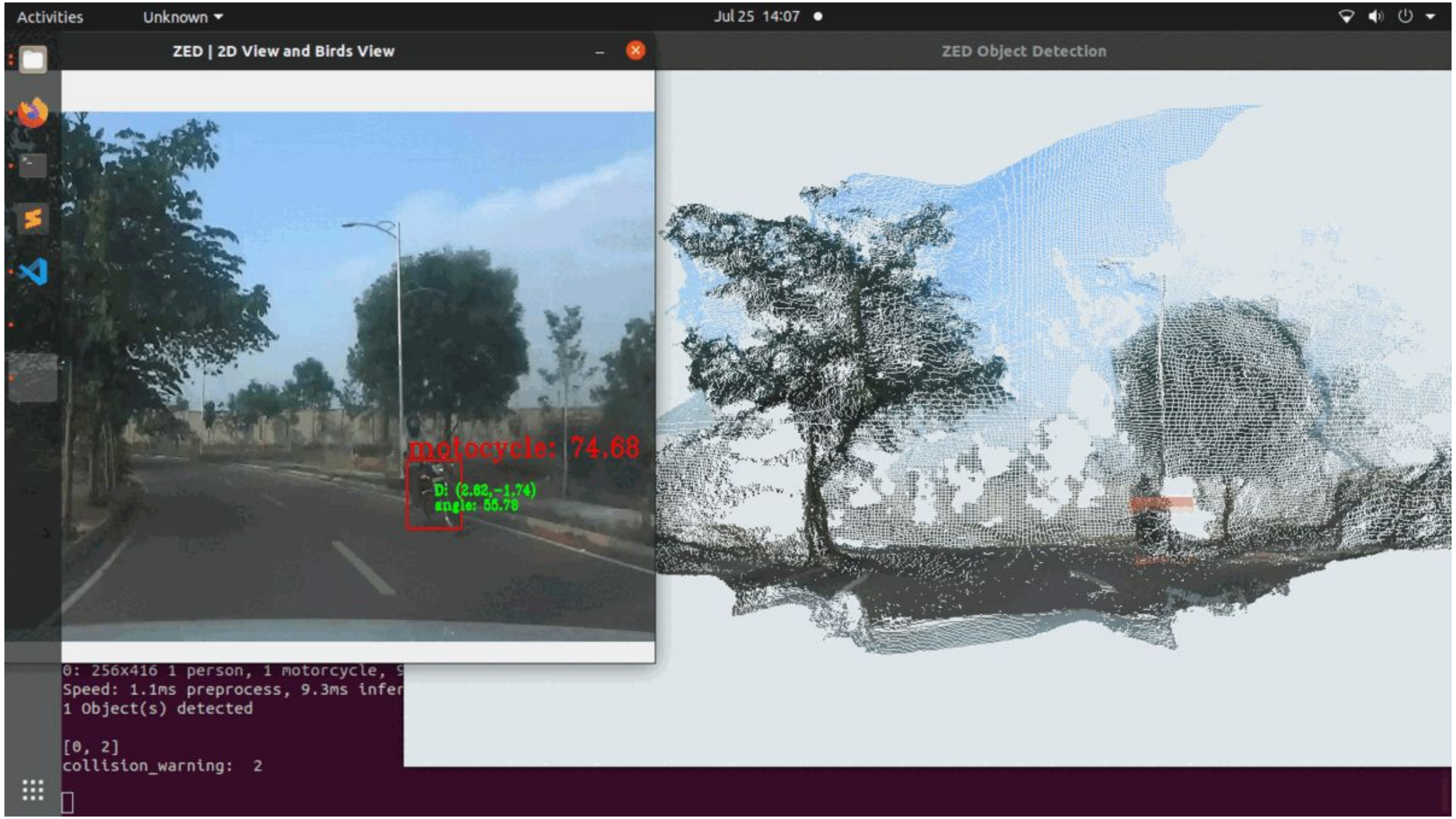


1 SV A = 0,0m (± 0,05 m)
2 vehicle path B = vsv = 30,0 km/h (± 0,25 km/h)
3 pedestrian path C = vtpL = 5,0 km/h (± 0,2 km/h)
4 virtual line on the outermost side of the pedestrian D ≥ 4,0 m
5 collision point E = Ld = 3,0 m (± 0,05 m)
L1 installed street lamp F = xc = 18,0m (± 0,25 m)
X from in

Link: https://drive.google.com/file/d/1x7D9FJeiqi1Qtj7PGz3om3WvkWbANgNX/view?usp=sharing

TiHAN-IIT Hyderabad
NextGen Autonomous Navigation

Collision Warning Module with YOLOv8 model



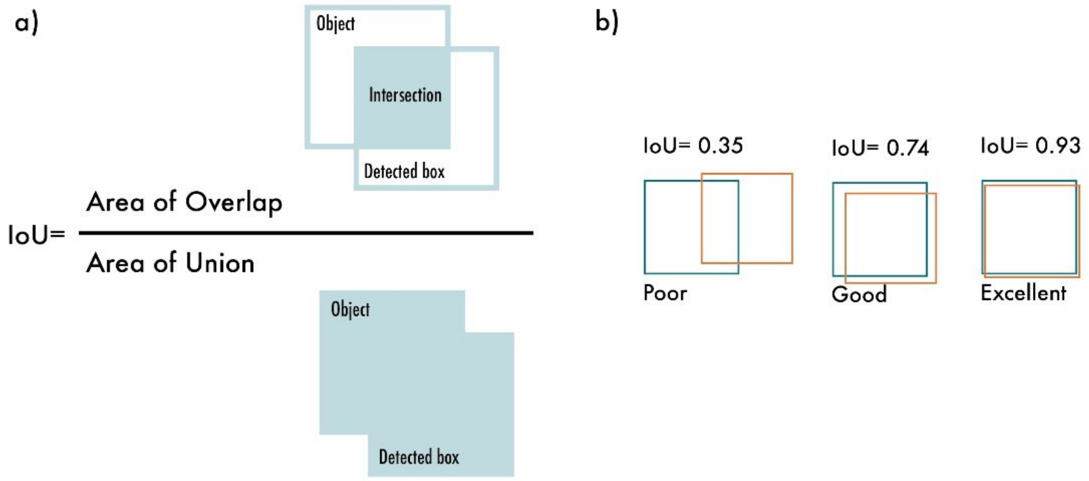
```
0: 256x416 1 person, 1 motorcycle,
Speed: 1.1ms preprocess, 9.3ms infer
1 Object(s) detected

[0, 2]
collision_warning:  2
```

CARLA predictions with bbox simulation

```
time taken for inference: 1.04ms
pre proc and nms: 0.58ms
[illegible] 22.36ms
```

AP and mAP in YOLO

- The Average Precision (AP), traditionally called Mean Average Precision (mAP), is the commonly used metric for evaluating the performance of object detection models. It measures the average precision across all categories, providing a single value to compare different models. The COCO dataset makes no distinction between AP and mAP.
- Intersection over Union: Object detection aims to accurately localize objects in images by predicting bounding boxes. The AP metric incorporates the Intersection over Union (IoU) measure to assess the quality of the predicted bounding boxes. IoU is the ratio of the intersection area to the union area of the predicted bounding box and the ground truth bounding box. It measures the overlap between the ground truth and predicted bounding boxes. The COCO benchmark considers multiple IoU thresholds to evaluate the model's performance at different levels of localization accuracy.
-



Evaluated on MS COCO dataset test-dev 2017, YOLOv8x achieved an AP of 53.9% with an image size of 640 pixels (compared to 50.7% of YOLOv5 on the same input size) with a speed of 280 FPS on an NVIDIA A100 and TensorRT.

Fig. Intersection over Union (IoU).
a) The IoU is calculated by dividing the intersection of the two boxes by the union of the boxes;
b) examples of three different IoU values for different box locations.