RBE/CS 549 Computer Vision

P3 - EinsteinVision

using 2 late days

Uthiralakshmi Sivaraman
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, MA, USA
usivaraman@wpi.edu

Noopur Koshta
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, MA, USA
nkoshta@wpi.edu

Sanya Gulati
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, MA, USA
sgulati@wpi.edu

***Abstract*—The project involves the reconstruction of an autonomous vehicle dashboard using Blender software and focuses on the challenges that arise in detecting and rendering obstacles using only images as sensors in autonomous vehicles. The dashboard aims to display real-time information related to the vehicle's surroundings, such as obstacles, traffic signals, and road conditions. The project involves the development of various scenarios to simulate different driving conditions and situations that an autonomous vehicle may encounter. The project's outcome will provide insights into the challenges faced in detecting and rendering obstacles using only images, which could lead to the development of new technologies and strategies to improve the performance of autonomous vehicles in real-world situations.**

***Index Terms*—Autonomous, Blender**

I. CHECKPOINT 1 : BASIC FEATURES

In this checkpoint, we'll be creating basic features in blender as follows:

- **Lanes:**
  Identify and show the different kinds of lanes on the road, they could be dashed, solid and/or of different color (white and yellow). Each lane has significance and are essential to identify.
- **Vehicles:**
  Here, identify all cars (but you do not need to classify as different types) and represent them as a car shape. Here we tried nuScenes dataset to train model to give car classes.
- **Pedestrians:**
  Identifying and locating pedestrians in the scene and depicting their poses.
- **Traffic lights:**
  Indicate the traffic signals and it's color alongwith arrows.
- **Road signs:**
  There are sign boards on the road you need identify and represent such ad Stop Sign and speed limits.

To identify the x and y positions of any of these objects, we used YOLOv5 and to get z-coordinate of objects, we created depth map. For each bounding box of object, centtroid was obtained and then the relative depth of that object pixel in that depth map is obtained and using those we obtained relative distance in world frame.

We use MiDAS to obtain monocular depth. MiDAS (Multiple intelligent Driving Agents for Safety) is a research project focused on the development of intelligent transportation systems for autonomous driving. The project is led by the University of Michigan and involves collaboration with a number of automotive industry partners.

MiDAS data refers to the data collected by the sensors and cameras used in the autonomous vehicles developed as part of the MiDAS project. This data is used to train machine learning algorithms and improve the performance of the autonomous driving system.

The MiDAS project aims to address the safety, efficiency, and environmental challenges associated with transportation through the development of autonomous driving technology. The project has been ongoing since 2015 and has produced a number of research papers and prototype autonomous vehicles.

MiDAS (Multiple intelligent Driving Agents for Safety) is an autonomous driving system that uses a geometric approach to estimate depth and build a 3D map of the environment around the vehicle. This is achieved by combining information from multiple sensors, including cameras and LiDAR (Light Detection and Ranging) sensors.

The basic idea behind the geometric approach is to use the geometry of the scene to estimate the depth of different objects. This involves finding correspondences between points in the 2D image captured by the camera and points in the 3D world. The process involves the following steps:

- Feature Detection: The first step is to detect feature points in the 2D image, such as corners or edges, that can be used to find correspondences with points in the 3D world.
- Feature Matching: The next step is to find correspondences between the feature points in the 2D image and the 3D world. This is done by comparing the descriptors

of the feature points in the 2D image with those in the 3D world.

- Estimation of Fundamental Matrix: Once the correspondences are found, the fundamental matrix is computed. The fundamental matrix relates the 2D image points with the 3D world points.
- Estimation of Essential Matrix: The essential matrix is then estimated from the fundamental matrix and the intrinsic parameters of the camera. The essential matrix relates the 2D image points with the 3D world points and also takes into account the camera parameters.
- Camera Pose Estimation: The camera pose is then estimated from the essential matrix. This gives the position and orientation of the camera in the 3D world.
- Triangulation: Finally, triangulation is used to estimate the depth of the points in the 3D world. This involves finding the intersection point of two rays that pass through the camera and the 3D world point.
  The formula for triangulation is given by:
  (x1 - cx) / fx = (X - Xc) / Zc
  (y1 - cy) / fy = (Y - Yc) / Zc
  where (x1, y1) are the pixel coordinates of the feature point in the 2D image, (cx, cy) are the coordinates of the principal point of the camera, fx and fy are the focal lengths of the camera, (Xc, Yc, Zc) are the coordinates of the camera in the 3D world, and (X, Y, Z) are the coordinates of the 3D world point.

By repeating this process for multiple points in the image and combining the results, MiDAS can build a detailed 3D map of the environment around the vehicle, which can be used to navigate and avoid obstacles in real-time.

A. Blender:

Blender is a powerful 3D graphics software that can be used to create stunning images and animations. One of the key features of Blender is its ability to create videos using rendered images.

Here are the key steps involved in using Blender images to create videos:

- **Storyboard:**
  The first step in creating a video using Blender images is to create a storyboard. A storyboard is a visual representation of the video that outlines the key scenes and camera angles. It is important to have a clear vision of the video before starting to create the images.

- **Scene Set-Up:**
  Once the storyboard is complete, the next step is to set up the scenes in Blender. This involves creating the 3D models, setting the lighting, and choosing the camera angles. It is important to ensure that the scenes are consistent with the storyboard and that they will render correctly.

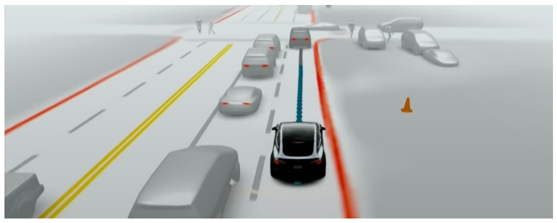

Fig. 1. A sample showing pedestrians, different vehicles, double yellow solid lines, traffic lights, stop sign and arrows on the road.

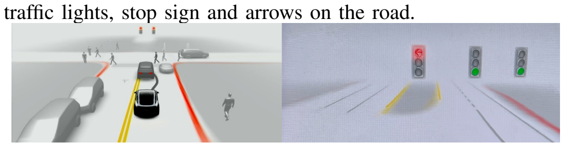

Fig. 2. Left: A visualization during a traffic stop, Right: Zoomed in view of traffic lights showing more details such as arrows and different colors (when applicable).

- **Image Rendering:**
  After the scenes are set up, the next step is to render the images. This involves using Blender to create high-quality images that will be used to create the video. It is important to render the images at a high resolution to ensure that they look good when combined to create the video.

- **Video Editing:**
  Once the images are rendered, the next step is to edit them together to create the final video. This involves using a video editing software or a python script to combine the images, add transitions, and add audio.

- **Exporting:**
  Finally, the video must be exported in a suitable format for distribution. This may involve choosing a suitable resolution, file format, and compression settings.

  Blender is a powerful tool that can be used to create high-quality images and videos. Using Blender images to create videos requires careful planning and attention to detail. By following the process outlined in this report, users can create professional-looking videos that showcase their creativity and technical skills.

II. CHECKPOINT 2 : ADVANCED FEATURES

For checkpoint 2, we had to introduce following features:

- **Vehicles:**
  Here, we need to classify (identify different vehicles) and subclassify them (identify different kinds of a type of vehicle). More particularly, Cars (Sedan, SUV, hatchback), Pickup trucks, Trucks, Bicycle, Motorcycle.
  For this, we trained networks on nuScenes dataset and Stanford car classification dataset, but those networks lacked accuracy in our data. Still ResNet-18

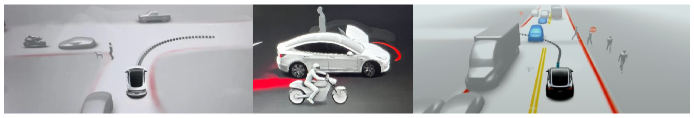

Fig. 3. Various scenarios showing different vehicles such as cars, trucks, motorcycles, etc.

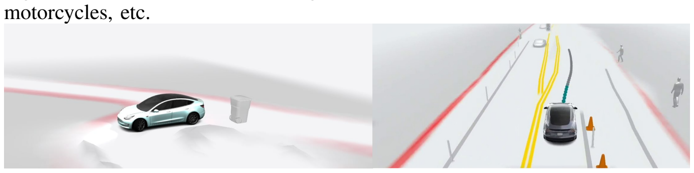

Fig. 4. Various scenarios showing different objects such as dustbins, traffic poles, traffic cones.

based pre-trained model performed fairly accurwith our data, with in-depth classes like brand and make of the car.

- **Traffic lights:**
  Additionally to the previous checkpoint, classify arrows on the traffic lights here.

- **Road signs:**
  Along with the previously mentioned sign boards, you should also indicate road signs on the ground such as arrows

- **Objects:**
  You also need to indicate additional objects like dustbins, traffic poles, traffic cones and traffic cylinders as their respective 3D models in the renders.

III. CHECKPOINT 3 : Bells and Whistles

- **Break lights and indicators of the other vehicles:** The detection and display of vehicle brake lights and indicator signals can assist the navigation module in making more informed decisions when changing lanes, ultimately improving driving safety and efficiency.

- **Pedestrian pose:** You need to identify pedestrian pose each frame instead of just classifying them and display them. For this we trained YOLO3D, which gave poses of cars and pedestrians. Also we tried classical approaches but merging them in Blender was again a challenges we faced here.

- **Parked and Moving Vehicles:** Distinguish between parked and moving vehicles and display (make it subtle but identifiable). For this, we tried classical optical flow methods, and the thought process was even that if the car observed is outside the lane contours then we can call it parked otherwise the vehiicles inside the lane are not. This would nee time to render, thus dropped the plan. Also, nuScenes dataset had pre-trained models to identify parked and moving vehicles.

IV. Process

- **Finding Networks:** The first step is to look for networks that identify the above listed objects. Using which, we obtain bounding boxes around those objects.

- **Obtaining Coordinates:** The 2D coordinates of the bounding boxes are stored in a csv file using pre-trained YOLOv5 model.

- **Obtaining Depth:** The depth must be known to place the coordinates in the world, since it represents z-axis. For this, we use MiDaS and obtain outputs of relative depths.

- **Scaling:** Since the outputs of different models are in pixels of images of different sizes and the depth is also relative, they need to be scaled together to be placed in the blender world.

- **Loading Objects:** The objects can be loaded using python script or modelled in the blender at specific location, rotation and orientation.

V. Networks used

- **Lanes:**
  - Network: YolovP2
  - Dataset trained on: BDD100K dataset
  - Command to obtain images with bounding boxes: $pythoncustom_demo.py - -source../input/input_image.jpg - -devicecpu - -namecpu_output$

- **Vehicles:**
  - Network:yolov3
  - Dataset trained on:COCO dataset
  - Command to obtain images with bounding boxes: $python3/home/sanya/Road - sign - detection/yolov5/detect.py - -weights/home/sanya/Road - sign - detection/Pretrained_weights/best.pt - -source/home/sanya/Desktop/CV/P3/P3Data/Sequence$

- **Pedestrians:**

- Network: YOLOv3
- Dataset trained on:COCO dataset
- Command to obtain images with bounding boxes: $python3/home/sanya/Road -$
  $sign - detection/yolov5/detect.py -$
  $-weights/home/sanya/Road - sign -$
  $detection/Pretrained_weights/best.pt -$
  $-source/home/sanya/Desktop/CV/P3/P3Data/Seque$

- **Traffic Lights:**
  - Network: yolov3
  - Dataset trained on:LISA Traffic Lights dataset
  - Command to obtain images with bounfing boxes: $pythondetect.py -$
    $-source/home/sanya/Desktop/CV/P3/P3Data/Seque$
    $03 - 03_15 - 31 - 56 - front.mp4 - -view - img -$
    $-weightsweights/best_model_12.pt - -img -$
    $size608$

- **Road Signs:**
  - Network: YOLOv5
  - Dataset trained on: Road Sign Detection, Kaggle
  - Command to obtain images with bounfing boxes: $python3/home/sanya/Road -$
    $sign - detection/yolov5/detect.py -$
    $-weights/home/sanya/Road - sign -$
    $detection/Pretrained_weights/best.pt -$
    $-source/home/sanya/Desktop/CV/P3/P3Data/Sequences/frames$

- **Depth Map:**
  - Network: MiDAS

VI. Conclusion

We found some great repositories but they are not renderable because it's difficult to extract bounding box coordinates. We also had to make sure that the csv files generating for all the coordinates were in similar formats to allow smooth rendering. Since output of several networks had to combined together into a single blender world, all of those networks generated output imaged in different sizes giving bounding boxes coordinates in relative pixel values, all of which had to be scaled to look uniform into a single blender world which was a challenging task.

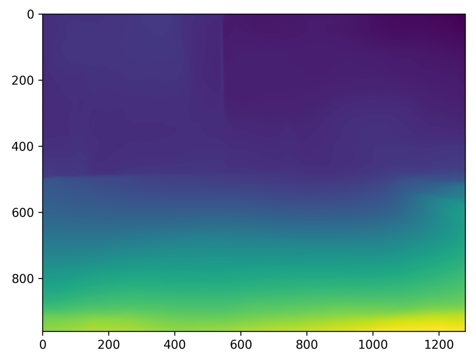

Fig. 5. Depth Frame

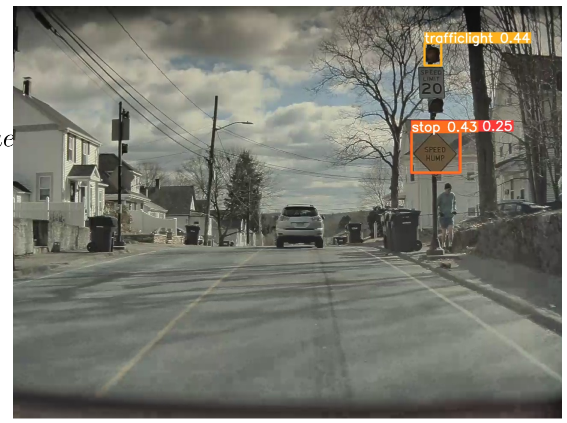

Fig. 6. Predicting Traffic light and Stop Sign

References

[1] https://github.com/Anant-mishra1729/Road-sign-detection

[2] https://github.com/sovit-123/Traffic-Light-Detection-Using-YOLOv3Get-the-Dataset

[3] https://pjreddie.com/darknet/yolo/

[4] https://www.geeksforgeeks.org/

[5] https://paperswithcode.com/

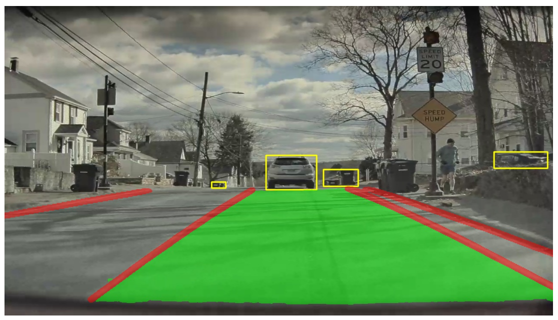

Fig. 7. Predicting Lanes

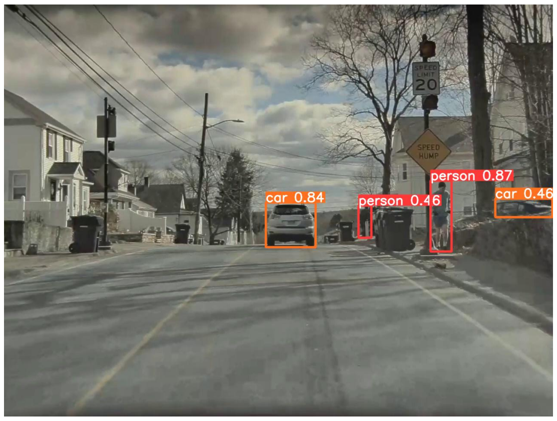

Fig. 8. Predicting Cars and Pedestrians

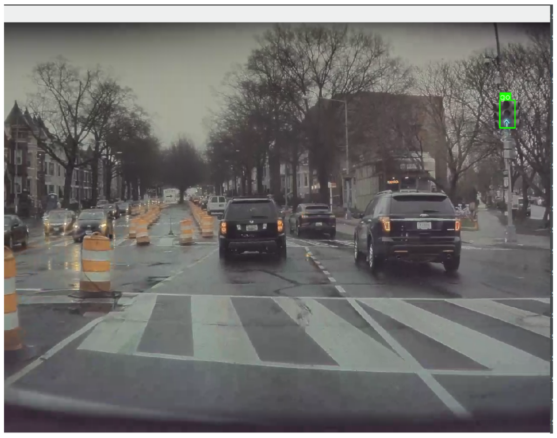

Fig. 9. Predicting Green Traffic Light

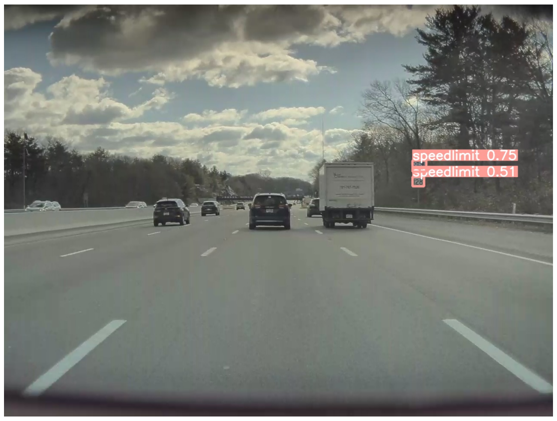

Fig. 10. Predicting Speed Limits

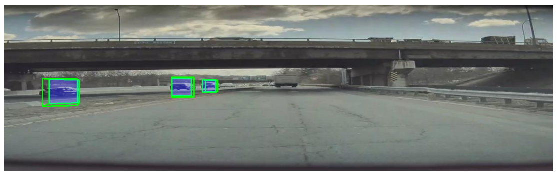

Fig. 11. 3D Bounding Box for Pose of Cars

Fig. 12. Rendered Image

[6] https://stackoverflow.com/

[7] https://github.com/

[8] https://colab.research.google.com/

[9]