Mapping and Perception for an Autonomous Robot
Project 4

Nadav Marciano 305165698

Part A: Visual Odometry

a. Visual odometry algorithm pipeline:

1. **Initialization:** Initializing the visual odometry system with the input data, including images and ground truth poses. Setting the initial camera pose to default values, typically an identity matrix for rotation and a zero vector for translation.

2. **Feature Extraction:** Using a feature extraction method to detect and describe key points in each image frame. Common methods include SIFT, SURF, or other feature detectors. Extracting key points and their corresponding descriptors from both the previous and current images.

3. **Feature Matching:** Matching the descriptors between consecutive frames to identify corresponding key points. This step involves using a matching algorithm like BFMatcher to find the best matches.

4. **Pose Estimation:** Computing the essential matrix or fundamental matrix based on the matched key points and the camera's intrinsic parameters. Deriving the rotation and translation between frames from the essential matrix to estimate the relative pose.

5. **Scale Estimation:** Determining the scale of the translation vector by comparing it with ground truth data. This step adjusts the translation to align with the true scale.

6. **Trajectory Calculation:** Integrating the estimated rotation and translation to update the camera's trajectory over time. Compare this trajectory with the ground truth to compute errors.

7. **Visualization:** Creating visualizations to display the estimated trajectory, key points, and error metrics. This can involve plotting graphs, saving images, or creating videos to review the results.

8. **Processing loop:** Processing on each frame in sequence, applying the steps above to extract features, match them, estimate the pose and update the trajectory.

This pipeline provides a comprehensive approach to visual odometry, ensuring accurate trajectory estimation and effective visualization of the results.

After running the algorithm on trajectory number 02, we obtained a good result. The estimated VO path closely matches the ground truth (GT), with a near-identical alignment. The Euclidean error is approximately 2.7 meters after 500 frames. This can be seen in Figure 1.

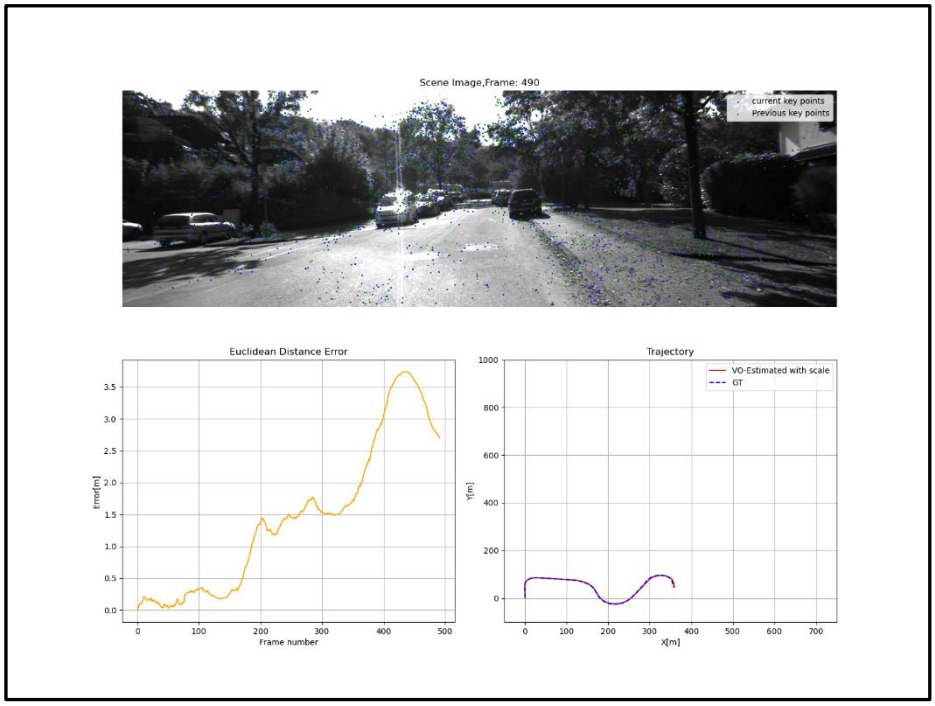


Figure 1 - Scene Image Frame 500, Trajectory (GT vs. VO), and Distance Error

b. Drift in visual odometry can arise from several factors:

- Feature matching errors can significantly impact visual odometry accuracy. Inaccurate key points detection, especially in cluttered or repetitive scenes, can result in a lack of distinctive key points, leading to poor matches and unreliable pose estimates. Additionally, incorrect matching of features between frames often due to similar looking elements in the scene can introduce large errors in the estimated trajectory, affecting the overall accuracy of the visual odometry system.

- Environmental changes can significantly affect visual odometry performance. Variations in lighting and weather conditions between frames can alter the appearance of features, leading to mismatches and drift in the trajectory. For example, shadows or reflections may disrupt feature detection and matching. Additionally, changes in the scene, such as the introduction of new objects or alterations in the layout, can impact feature consistency and pose estimation, further contributing to inaccuracies in the estimated trajectory.

- The presence of moving objects, such as pedestrians, vehicles, or animals, can introduce noise into feature tracking, causing inaccuracies in pose estimation and resulting in drift. Additionally, interference from background movement, like swaying trees or passing traffic, can disrupt feature stability and further contribute to errors in the trajectory estimation.

The comparison between the estimated visual odometry (VO) trajectory and the ground truth (GT) trajectory reveals that, while the paths are generally similar, there are noticeable discrepancies. The VO path closely follows the GT path with some deviations, particularly in areas with complex features or high motion. Some examples of drift:

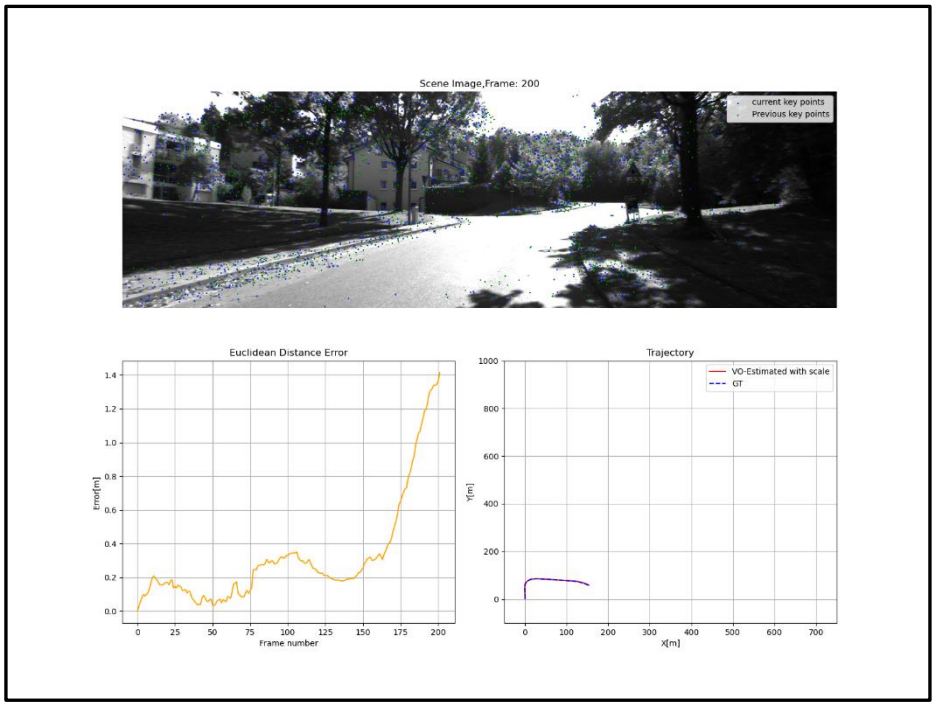


Figure 2 - Scene Image Frame 200, Trajectory (GT vs. VO), and Distance Error

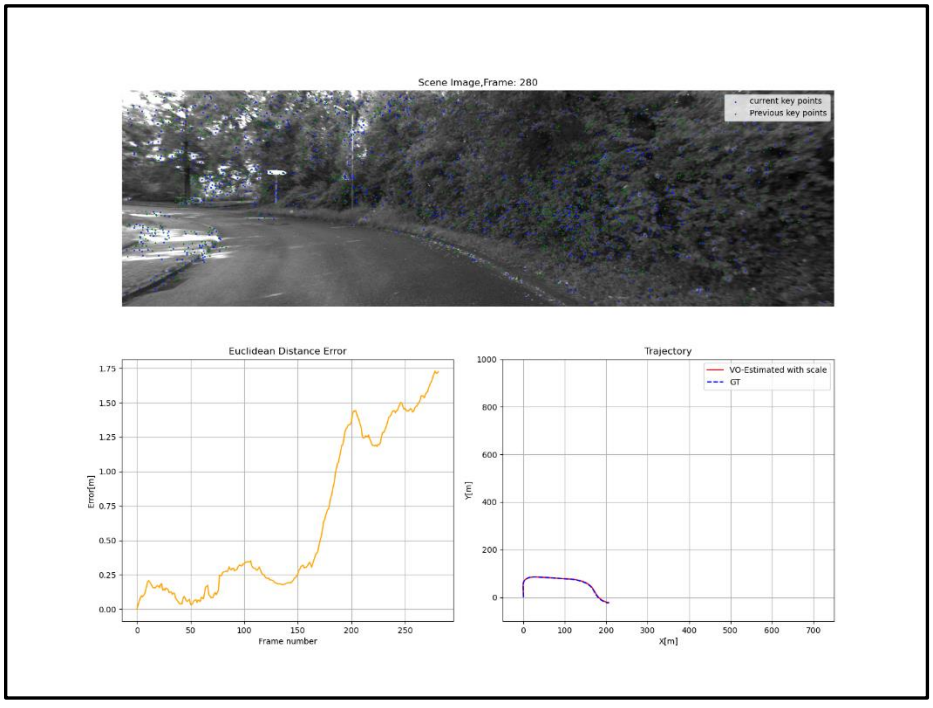


Figure 3 - Scene Image Frame 280, Trajectory (GT vs. VO), and Distance Error

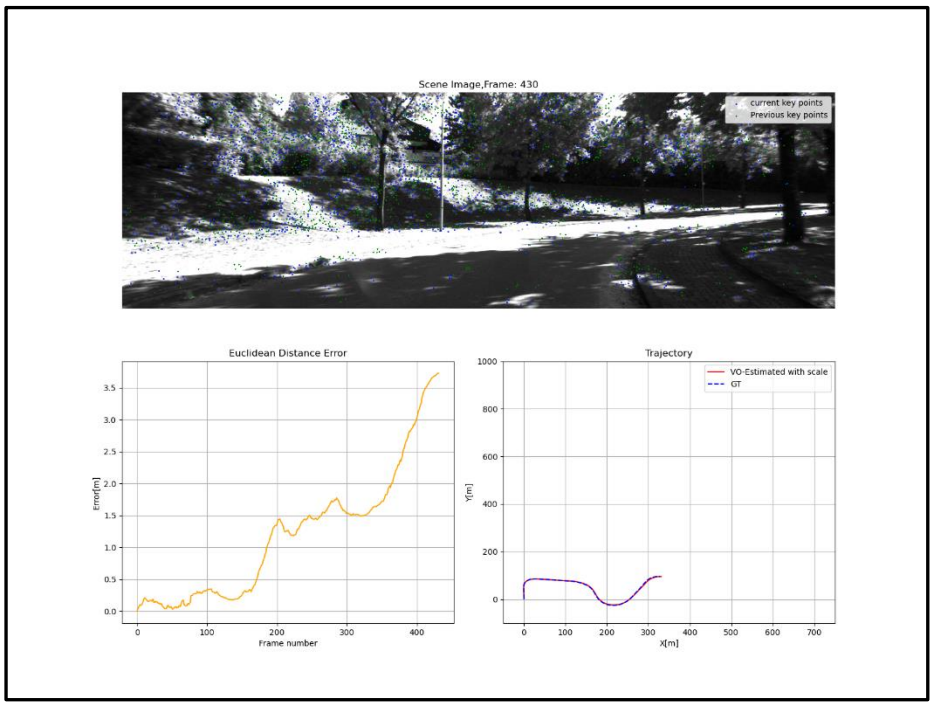


Figure 4 - Scene Image Frame 430, Trajectory (GT vs. VO), and Distance Error

In these examples, we observe significant drift due to changes in the scene, including vehicle turns and varying lighting conditions. Although the image appears black and white image, sunlight reflections on the road complicate the algorithm's ability to find matching points. Additionally, dynamic elements like moving vehicles, present during the scene, contribute to the drift. The combination of these factors scene changes and dynamic objects—primarily impacts the accuracy of trajectory estimation.

To reduce drift in visual odometry results, the following improvements can be implemented:

- Integration of data from additional sensors, such as IMU or GPS, to provide supplementary information and reduce dependence on visual data alone. Combining measurements from these sensors can enhance the accuracy of pose estimation by incorporating multiple sources of information.

- Using sensor fusion techniques, such as Kalman Filters or Extended Kalman Filters, to effectively combine visual data with measurements from additional sensors. These filters help in improving pose estimation by merging different types of sensor data, which can lead to a more robust and accurate trajectory estimation.

- Implementation of an algorithms that can handle variations in lighting, weather conditions, and scene changes. Techniques such as adaptive thresholding for feature detection or scene normalization approaches can help mitigate the impact of these environmental factors on the visual odometry system.

c. For an autonomous tractor in a sunflower crop, I would use **SURF** for feature extraction. Sunflower fields have repetitive patterns, like rows of sunflowers. SURF can handle these patterns relatively well because it’s designed to detect and describe features that are invariant to changes in scale, rotation, and lighting. Even though the texture might be repetitive, SURF can still find

distinctive points, like the tops of sunflower heads or gaps between plants. The repetitive nature of a sunflower field might pose a challenge for any feature detection algorithm, including SURF. If the sunflowers are too uniform, the algorithm might struggle to find unique features, which could make matching less reliable. However, if there are enough variations (e.g., differences in plant size, spacing, or gaps), SURF can still identify distinct points to match across images.

For a car in an urban environment, I'd choose **SIFT** for feature extraction. Urban areas have a lot of different features like buildings, road signs, dynamic objects such as moving cars, and pedestrians. SIFT is great at finding and matching these features, even when the scale or angle changes. It's a bit slower but very accurate, which is important for navigating a busy city with many moving and static elements. In planning visual odometry for a car driving through urban scenes there are many challenges, two specific challenges are:

- **Dynamic Objects:** Urban environments are full of moving objects like cars, pedestrians, and cyclists. These dynamic elements can complicate visual odometry because the algorithm needs to differentiate between static features (like buildings and road signs) and moving ones. Misidentifying moving objects as reference points can lead to inaccurate estimates of the car's position and movement.

- **Complex Lighting Conditions:** Urban scenes often have inconsistent lighting due to shadows from buildings, changing weather, and various artificial lights (like streetlights and headlights). These lighting changes can alter the appearance of features in the images, making it more difficult for the algorithm to detect and match features reliably across different frames. Managing these lighting variations is essential for accurate visual odometry in such environments.

Part B: 3D object detection

a. The Feature Encoder, known as the Pillar Feature Net, is responsible for converting a 3D point cloud into a sparse pseudo-image as part of the backbone of the PointPillars network.

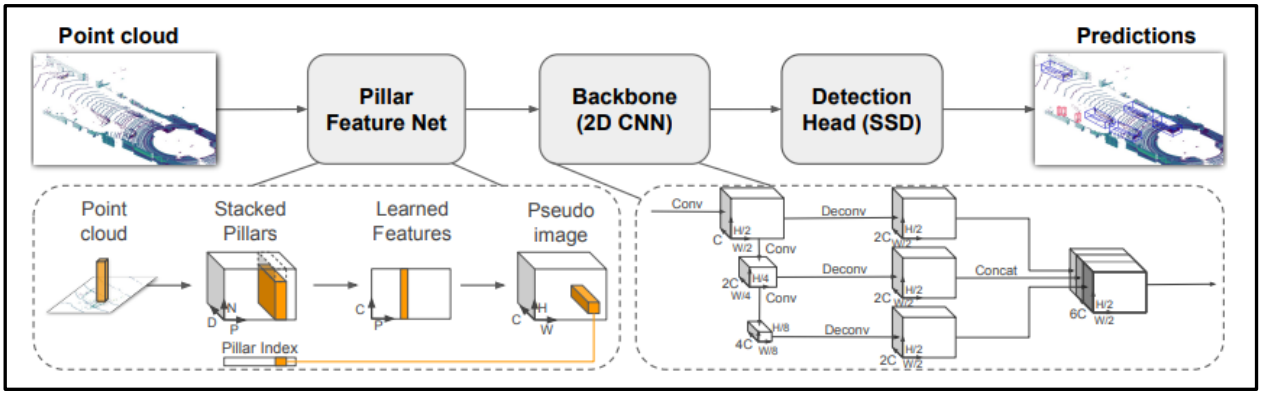


Figure 5 - PointPillars network overview

Each component of the network plays a crucial role in refining this data and generating accurate 3D object detection. Below is a breakdown of how the Pillar Feature Net converts the 3D point cloud into a sparse pseudo-image:

- **Pillar Feature Net (Feature Encoder):** The Pillar Feature Net is the first stage where the 3D point cloud is processed. The point cloud is discretized into an evenly spaced grid in the x-y

plane, forming a set of "pillars." Each pillar is a vertical column in the grid containing all points within its spatial boundaries. For each point, additional features like distances from the pillar center and the mean of points in the pillar are calculated, resulting in an augmented feature vector. This augmentation helps in capturing both the local geometry and the relative position of the points within each pillar. The resulting feature vectors are then used to populate a pseudo-image, where each "pixel" corresponds to a pillar's aggregated features, representing the spatial structure of the scene.

- **Backbone (2D CNN):** The pseudo-image is then processed by the Backbone network, typically a 2D Convolutional Neural Network (CNN). This network extracts spatial features by applying convolutional filters across the pseudo-image, capturing patterns like edges, corners, and shapes that are critical for object detection.

- **Detection Head (SSD):** After processing by the Backbone, the Detection Head, often utilizing a Single Shot MultiBox Detector (SSD), takes the extracted features and predicts the properties of potential objects. The Detection Head generates 3D bounding boxes, rotation angles, and confidence scores for each detected object. These predictions are made across multiple scales and positions, ensuring accurate detection of objects with varying sizes and orientations.

- **Predictions:** The final output of the network includes detailed information for each detected object, such as its 3D position, dimensions, orientation, and confidence score. This data allows for accurate reconstruction and recognition of objects in the 3D environment.

These components work together to convert raw 3D point cloud data into a structured pseudo-image, making it possible to apply efficient 2D CNN techniques for 3D object detection. The resulting predictions provide a comprehensive understanding of the 3D scene, accurately identifying and localizing objects in the environment.

b. Example 1:

Figure 6 - Frame example 1

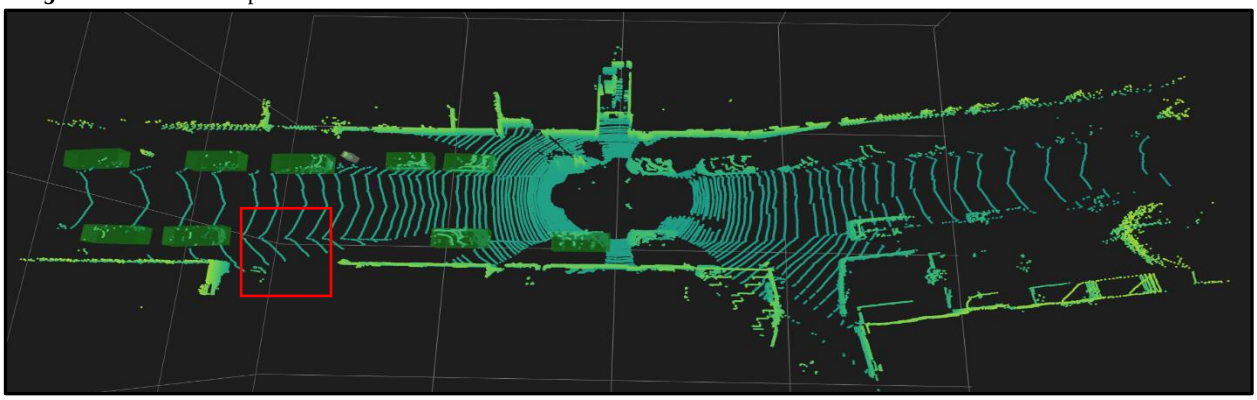
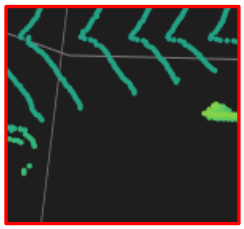


Figure 7 - 3D object detection points cloud representation of example 1

The scene depicts several parked cars along a narrow residential street. No dynamic objects are visible on the road, except for a pedestrian on the left side at the end of the street. We will attempt to visually identify a few key points to better understand the angle.
The detection results are generally quite good, although there is one instance of failure. The model successfully detects most of the parked cars and encases them within 3D bounding boxes.



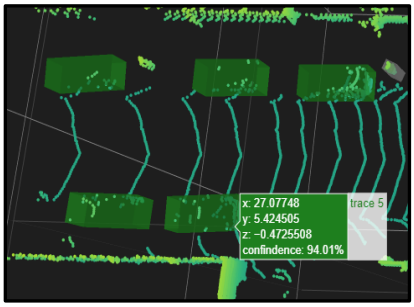
There is a wide gap between the first parked car on the left and the second car further down the street. This gap aids in better understanding the scene and accurately positioning the detected objects.

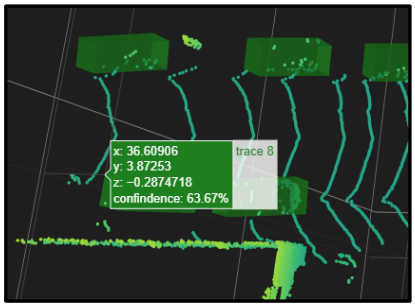
Confidence Table:

Trace number	Class	Confidence
1	Pedestrian	41.20 %
2	Car	96.99 %
3	Car	95.38 %
4	Car	94.63 %
5	Car	94.01 %
6	Car	92.92 %
7	Car	87.23 %
8	Car	63.67 %
9	Car	50.56 %
10	Car	49.58 %

Table 1 – Confidence scores for object detection of example 1

Each detection is associated with a confidence score, which tends to be higher for cars that are closer and less occluded. These cars are generally represented by a greater number of 3D points. For a detailed comparison, consider Trace 5 versus Trace 8:





Trace 5 has more 3D points and is less occluded, resulting in a significantly higher confidence score of 94.01%, compared to 63.67% for Trace 8. As the distance increases, the confidence values typically decrease, as observed between Trace 2, Trace 3, and the lower confidence of Trace 5.

Classification Failure:

Figure 8 – Frame example 1

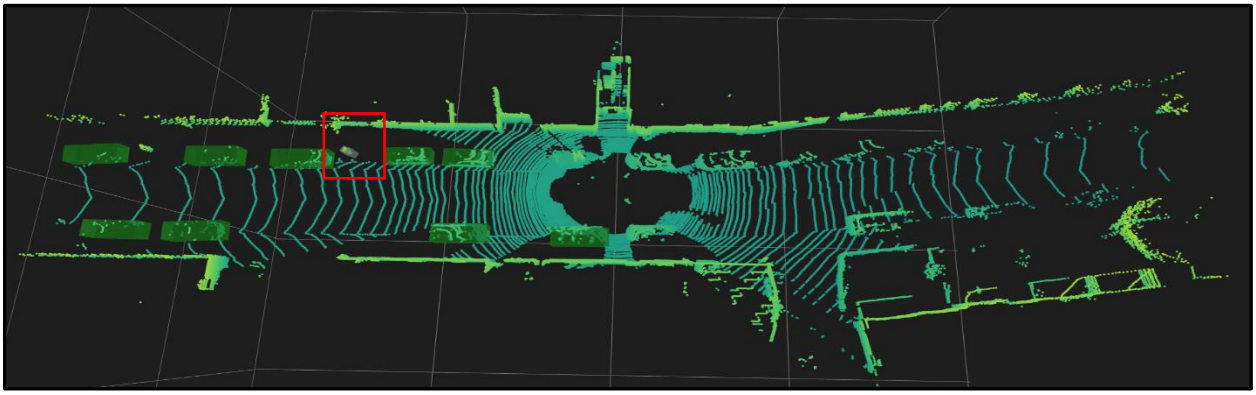


Figure 9 – 3D object detection points cloud representation of example 1

In addition to the parked cars, we also identify a tree in the point cloud. However, this tree was incorrectly classified as a pedestrian. This misclassification likely occurred because the tree's height was not accurately captured. The LiDAR seems to have detected only part of the tree, leading to an incorrect estimation of its size.
Notably, the low confidence value associated with this detection suggests that setting a confidence threshold could help filter out such misclassifications.

Example 2:

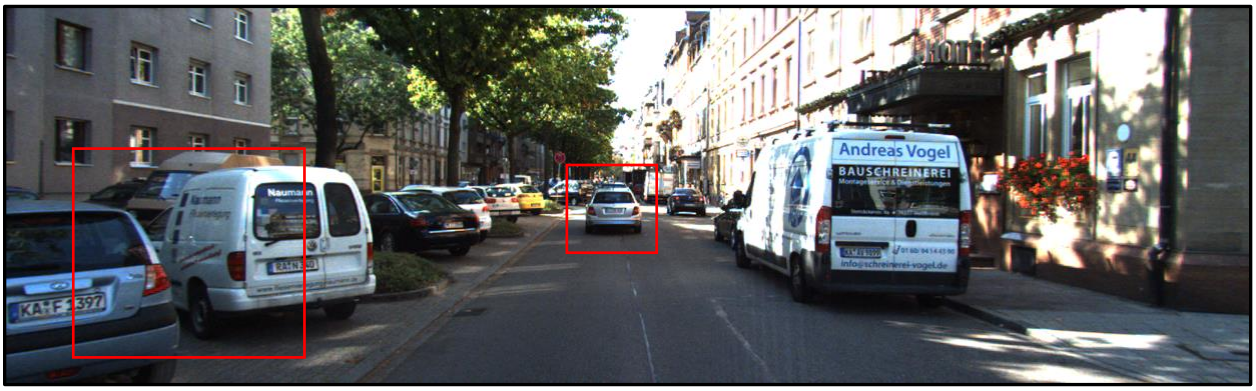


Figure 10 – Frame example 2

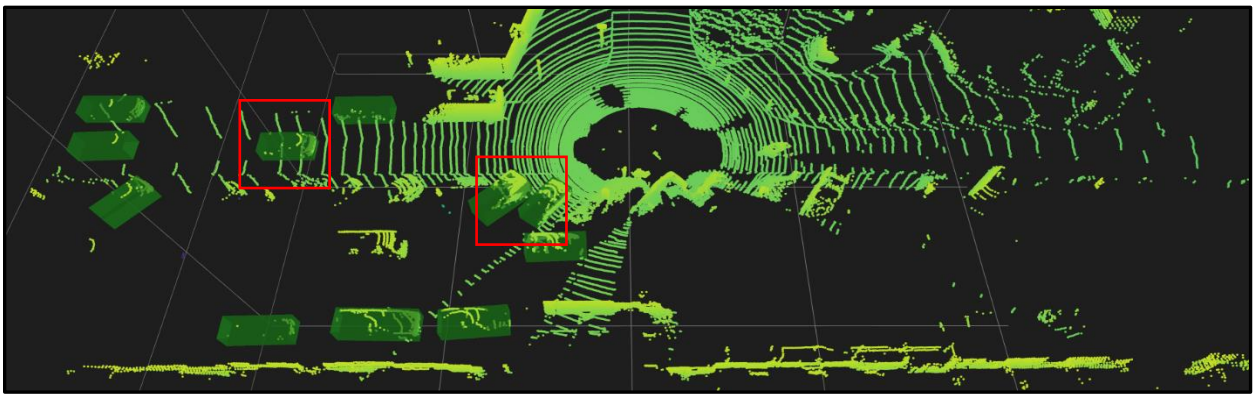


Figure 11 – 3D object detection points cloud representation of example 2

The scene depicts a main street with a one-way road and several lanes. Parked cars are present on both sides of the road, and there are dynamic objects, including moving vehicles. We will attempt to visually identify several key points to better understand the angle.

The vehicle traveling in the center of the road and the two parked cars on the left side, positioned diagonally, help in understanding the scene and accurately positioning the detected objects.

Confidence Table:

Trace number	Class	Confidence
1	Car	96.27 %
2	Car	95.24 %
3	Car	93.79 %
4	Car	93.66 %
5	Car	93.37 %
6	Car	92.43 %
7	Car	85.46 %
8	Car	80.66 %
9	Car	77.98 %
10	Car	74.89 %
11	Car	74.32 %
12	Car	72.79 %

Table 2 – Confidence scores for object detection of example 2

The model successfully located the first two parked cars on the left with at least 85% confidence, as they are relatively easy to recognize due to their diagonal parking. However, several other parked cars, despite having a significant number of points, were not detected at all.

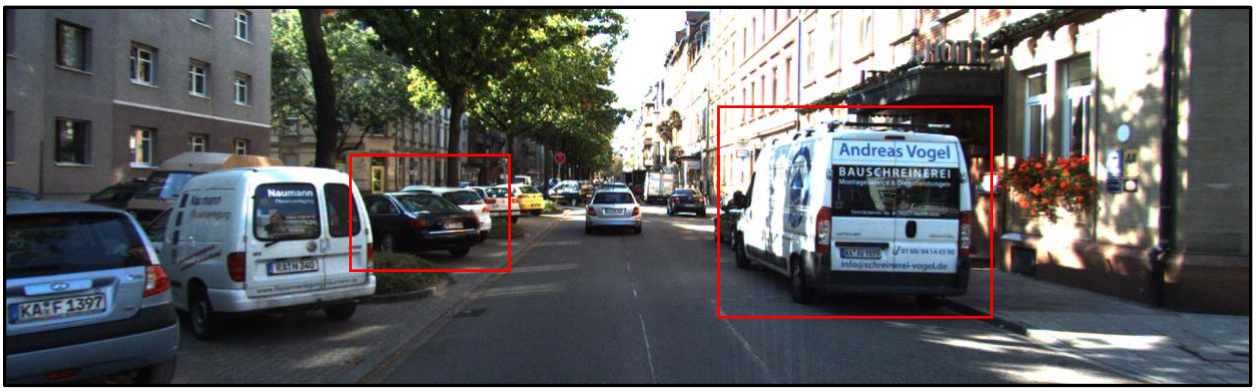


Figure 12 – Frame example 2

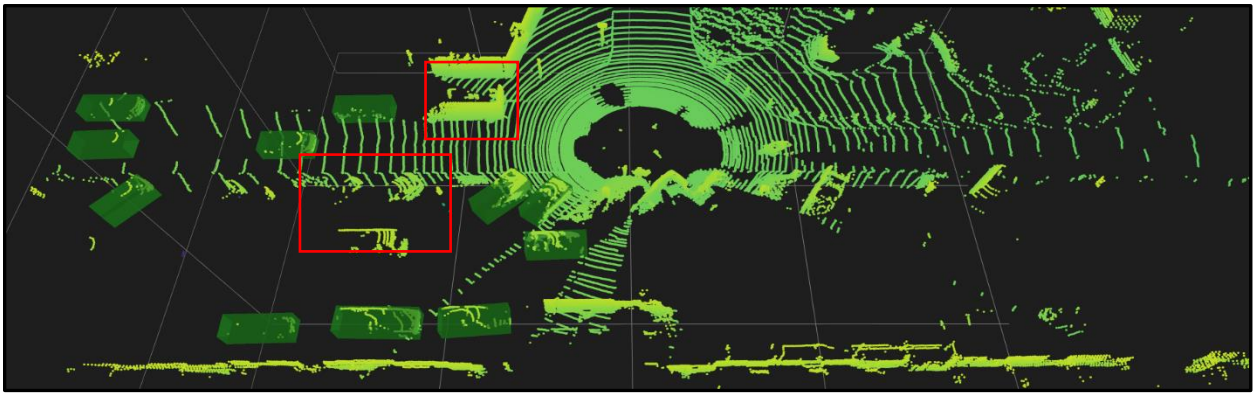


Figure 13 – 3D object detection points cloud representation of example 2

Additionally, the van on the right side was not detected. This is surprising given its proximity to the LiDAR and the substantial number of points it contains. It is possible that the size of the van contributed to the issue, the model's size thresholds for detecting vehicles may have excluded the van. This could be due to the specific angle from the LiDAR or the shape of the 3D point clouds affecting detection.

To include a van class in the PointPillars model, we need to update the anchor sizes to match the typical dimensions of a van, ensuring accurate bounding box predictions. Additionally, the model's class labels must be modified to include the van as a distinct category. Lastly, the training data should be adjusted to incorporate annotated examples of vans, allowing the model to learn and accurately detect them during inference.

Example 3:

Figure 14 – Frame example 3

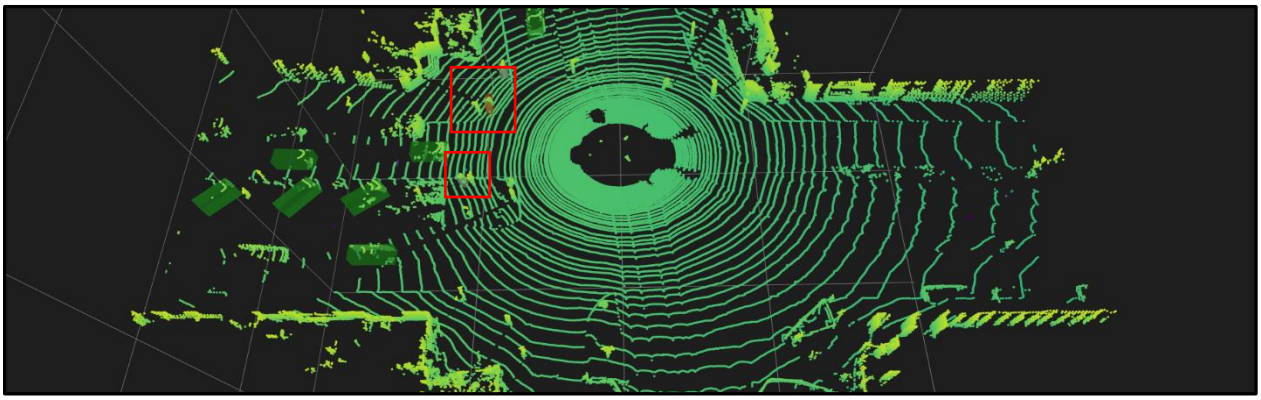


Figure 15 – 3D object detection points cloud representation of example 3

The scene depicts a main street with a two-way road, featuring parked cars on both sides. Additionally, there are dynamic objects including pedestrians, cyclists, and moving vehicles. We will attempt to visually identify several key points to better understand the angle.

The traffic sign and the cyclist are key elements for understanding the scene and accurately positioning the detected objects.

Confidence Table:

Trace number	Class	Confidence
1	Pedestrian	85.46 %
2	Pedestrian	55.90 %
3	Cyclist	94.37 %
4	Car	96.76 %

5	Car	93.71 %
6	Car	90.87 %
7	Car	89.62 %
8	car	88.18 %
9	car	84.81 %
10	car	81.48 %
11	car	72.56 %
12	car	66.28 %

Table 3 – Confidence scores for object detection of example 3

Despite the complex scenario with numerous static and dynamic elements, we observe a clear visualization of both the traffic sign and the cyclist in the point cloud. The cyclist is detected with high confidence, approximately 94%, while the pedestrian is detected with around 85% confidence.
As we know, the model detects objects based on learned features from the pseudo-image, which is generated from pillars that reflect the structure of each object. Because of this, if a van is built differently from a car, it may be overlooked or ignored by the model. This occurs because the model doesn't treat each point in the cloud as an isolated parameter; instead, it considers the point in relation to its nearest neighbors. To address this issue, it's necessary to introduce van examples into the model's training process.

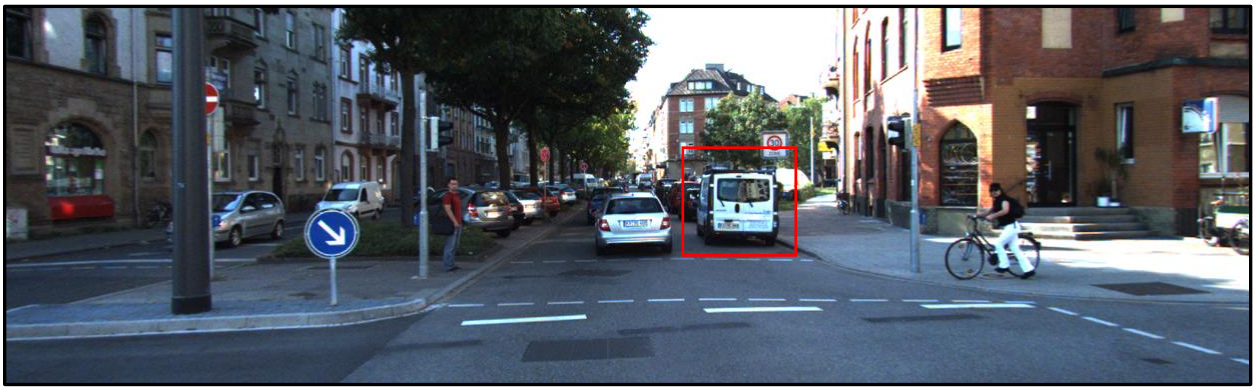


Figure 16 – Frame example 3

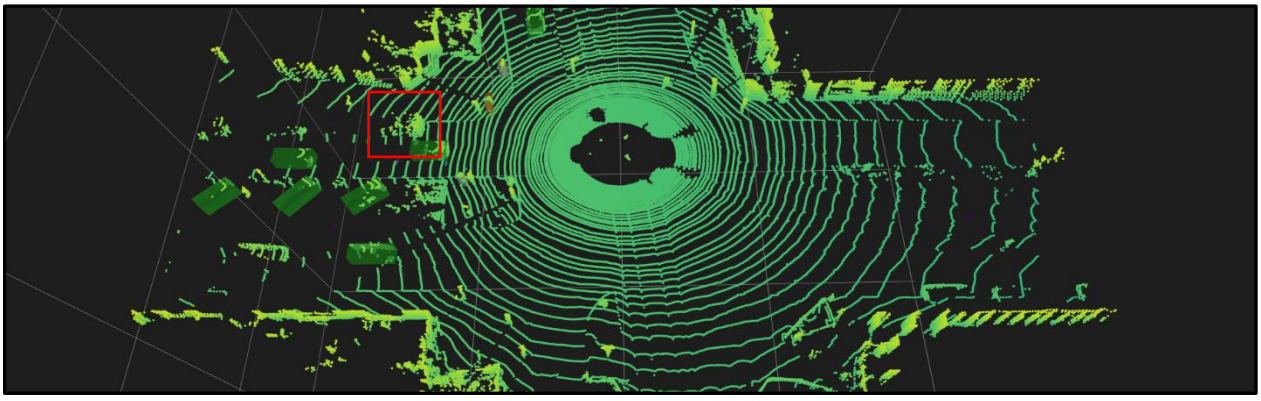
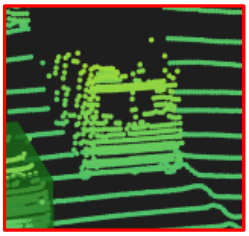


Figure 17 – 3D object detection points cloud representation of example 3



In this scene, we successfully detect all classes—car, cyclist, and pedestrian. However, there is a false positive where a pole (apparently) is misidentified as a pedestrian. We encountered a similar issue in Example A, and here again, the confidence is quite low, around 55%.

Figure 18 – Frame example 3

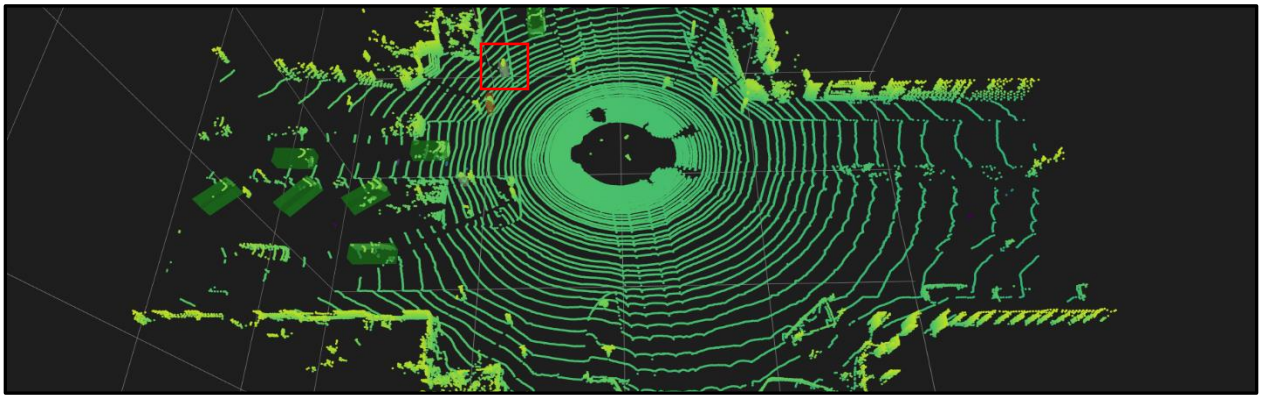
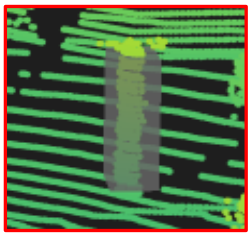


Figure 19 – 3D object detection points cloud representation of example 3



Several approaches could address this problem:

- **Data Augmentation:** A straightforward solution is to add more data and examples of pedestrians, which could help correct the model's tendency toward misclassification.

- **Regularization Techniques:** To improve detection, we can incorporate regularization techniques into the network's loss function, such as dropout or L1/L2 regularization. These methods help prevent overfitting and enhance the neural network's generalization ability. Overfitting can lead to false positives, as the network might memorize noise or outliers in the training data. Regularization can help mitigate this issue.

Part C: Multi Object Tracking

a. Multi-object tracking is a critical area of computer vision, where the goal is to accurately identify and follow multiple objects across a sequence of frames, despite challenges such as occlusions, varying object appearances, and complex motions.

BoT-SORT introduces several key improvements over ByteTrack, enhancing its effectiveness in multi-object tracking:

- Direct Bounding Box Estimation with Kalman Filter: Unlike ByteTrack, which uses a ratio to estimate bounding box size, BoT-SORT directly estimates the width and height of the bounding box using a Kalman Filter. This approach improves the accuracy of object tracking by better capturing the true size of objects as they move through the scene.

- Camera Motion Compensation: BoT-SORT addresses the challenges of dynamic camera movement by introducing Camera Motion Compensation. By calculating the affine transformation matrix between consecutive frames, the algorithm compensates for camera motion, reducing the likelihood of ID switches and false negatives. This ensures that the bounding box predictions remain accurate even when the camera is in motion.

- IoU-ReID Fusion: BoT-SORT enhances the association between detections and tracklets through IoU-ReID fusion. This method combines motion information (IoU distance) with appearance information (cosine distance), creating a more robust and reliable tracking system. By effectively merging these two aspects, BoT-SORT achieves better performance in maintaining consistent object identities across frames.

After running the BoT-SORT tracker on the five subsets from MOT17, the resulting data was processed to generate both the evaluation report and the corresponding animation results. These outputs were then saved, providing a comprehensive visual and quantitative analysis of the tracker's performance across the different subsets.

Comparison of Tracker Performance Across Sequences:

HOTA: BoTSORT-pedestrian	HOTA	DetA	AssA	DetRe	DetPr	AssRe	AssPr	LocA	OWTA	HOTA(0)	LocA(0)	HOTALocA(0)
MOT17-02	49.143	49.827	49.256	55.155	75.728	53.975	74.952	83.839	51.932	60.622	77.968	47.266
MOT17-04	79.173	78.162	80.544	83.774	86.15	84.602	88.06	88.628	82.104	90.362	86.999	78.614
MOT17-09	64.288	71.125	58.197	73.917	86.827	65.537	74.36	86.945	65.586	75.353	85.147	64.161
MOT17-10	59.348	57.79	61.259	61.973	76.961	65.327	78.782	81.422	61.594	77.063	77.393	59.641
MOT17-13	70.1	66.456	74.098	70.747	81.384	79.194	83.39	84.129	72.392	87.089	80.941	70.491
COMBINED	69.643	67.863	72.042	73.304	82.945	76.549	84.372	86.603	72.586	81.962	83.76	68.652

CLEAR: BoTSORT-pedestrian	MOTA	MOTP	MODA	CLR_Re	CLR_Pr	MTR	PTR	MLR	sMOTA	CLR_TP	CLR_FN	CLR_FP	IDSW	MT	PT	ML	Frag
MOT17-02	55.594	81.971	56.35	64.592	88.684	35.849	45.283	18.868	43.948	6403	3510	817	75	19	24	10	166
MOT17-04	89.879	87.31	89.969	93.606	96.26	89.855	7.2464	2.8986	78	22677	1549	881	22	62	5	2	92
MOT17-09	83.506	84.888	84.025	84.578	99.35	72.727	22.727	4.5455	70.725	2446	446	16	15	16	5	1	25
MOT17-10	73.512	78.293	73.798	77.161	95.823	44.444	50	5.5556	56.762	4588	1358	200	17	16	18	2	77
MOT17-13	81.827	81.743	82.016	84.472	97.174	70.455	18.182	11.364	66.405	2682	493	78	6	31	8	5	18
COMBINED	79.453	84.825	79.745	84.061	95.116	64.286	26.786	8.9286	66.696	38796	7356	1992	135	144	60	20	378

Identity: BoTSORT-pedestrian	IDF1	IDR	IDP	IDTP	IDFN	IDFP
MOT17-02	57.76	49.914	68.532	4948	4965	2272
MOT17-04	90.83	89.577	92.117	21701	2525	1857
MOT17-09	76.018	70.367	82.656	2035	857	427
MOT17-10	80.473	72.637	90.205	4319	1627	469
MOT17-13	89.368	83.528	96.087	2652	523	108
COMBINED	82.022	77.256	87.415	35655	10497	5133

VACE: BoTSORT-pedestrian	SFDA	ATA
MOT17-02	62.95	29.196
MOT17-04	83.202	70.457
MOT17-09	78.378	61.571
MOT17-10	67.768	56.729
MOT17-13	73.155	69.698
COMBINED	74.175	55.719

Count: BoTSORT-pedestrian	Dets	GT_Dets	IDs	GT_IDs
MOT17-02	7220	9913	102	53
MOT17-04	23558	24226	99	69
MOT17-09	2462	2892	23	22
MOT17-10	4788	5946	49	36
MOT17-13	2760	3175	48	44
COMBINED	40788	46152	321	224

Table 4 – MOT Evaluation Metrics

- MOTA: The tracker achieves the highest MOTA score in the MOT17-04 sequence (89.879), indicating excellent overall tracking accuracy in terms of both detection and association. The MOT17-02 sequence, on the other hand, has the lowest MOTA score (55.594), reflecting challenges in maintaining accurate tracking.

- MOTP: The tracker shows consistently strong precision across sequences, with MOT17-04 again leading (87.31), which suggests that the bounding boxes are very accurate in this sequence. Lower MOTP in MOT17-02 (81.971) indicates slightly less precision in bounding box localization.

Figure 20 – MOT17-04 with the highest performance metrics frame 358

- The HOTA score, which balances detection accuracy and association accuracy, is highest in the MOT17-04 sequence (79.173). This suggests that BoT-SORT is highly effective in this sequence in both detecting objects and maintaining consistent tracks over time.
  The lowest HOTA score is in the MOT17-02 sequence (49.143), highlighting the tracker's difficulty in maintaining both high detection and association accuracy in more challenging conditions.

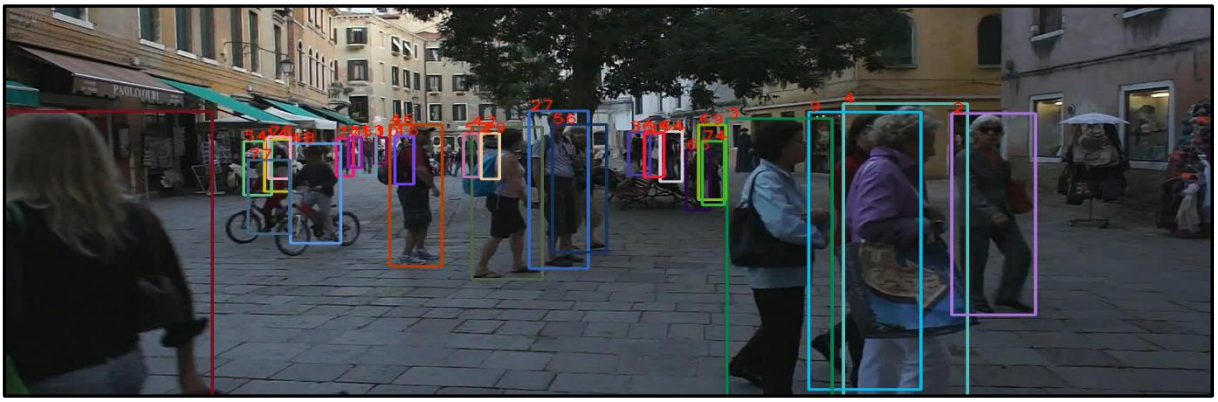


Figure 21 – MOT17-02 with the lowest performance metric frame 175

- The number of ID switches is a crucial metric in evaluating tracking performance. MOT17-02 exhibits the highest number of ID switches (75), suggesting frequent loss of object tracking and reassignment of IDs, which results in tracking inconsistencies. On the other hand, MOT17-13, which features fewer people in the scene, has the lowest number of ID switches (6). This lower count indicates better tracking consistency and stability, although it may be influenced by the simpler scene with fewer objects to track.

Figure 22 – MOT17-13 with the fewest ID switches frame 162

- IDF1: The highest IDF1 score is in MOT17-04 (90.83), demonstrating the tracker's strong capability to correctly identify and re-identify objects over the sequence. This is supported by the high IDR (89.577) and IDP (92.117) scores, showing that both recall and precision in identity matching are excellent. The lowest IDF1 score is in MOT17-02 (57.76), indicating significant challenges in correctly identifying objects throughout the sequence, likely due to frequent occlusions or complex object interactions.

- DetA: Detection accuracy is highest in MOT17-04 (78.162), indicating effective and precise object detection. The tracker performs well in detecting objects accurately across most frames in this sequence. DetRe and DetPr: MOT17-04 again shows the highest detection recall (83.774) and precision (86.15), suggesting that the tracker not only detects most of the objects but also does so with high precision. MOT17-02, however, shows the lowest recall (55.155) and precision (75.728), reflecting difficulties in detecting objects consistently and accurately.

Summary

The BoT-SORT tracker excels in the MOT17-04 sequence, showing high scores across all metrics (MOTA, MOTP, HOTA, IDS, Identity, and Detection Performance). This high performance is attributed to the advantageous camera angle from above, which simplifies object detection and tracking, even though the objects are dynamic. In contrast, the tracker struggles the most in MOT17-02. The lower scores in this sequence suggest challenges due to more complex environments, leading to less accurate tracking, higher ID switches, and poorer detection performance.

Reasons for Differences in Results

The differences in results can be attributed to the distinct characteristics of each sequence. MOT17-04 features a less crowded scene with fewer occlusions, allowing the tracker to maintain accurate associations and achieve higher detection and identification precision. Conversely, MOT17-02 presents more challenging conditions, including heavy occlusions and a camera angle directly facing the objects. This setup results in complex interactions and frequent object blockages, making it harder for the tracker to maintain accurate object identities. Consequently, this leads to a higher number of ID switches and lower overall accuracy.

b. Analysis of results:

examples of good crossing objects that keep the same ID:

Figure 23 – MOT17-09 frame 137

Figure 24 – MOT17-09 frame 158

In this scene, object ID 14 a woman, is being tracked while another woman, ID 12, crosses behind her. Despite the close proximity and potential for confusion, the model successfully maintains the correct IDs for both individuals after they cross paths.

Figure 25 – MOT17-10 frame 50

Figure 26 – MOT17-10 frame 66

In this scene, object ID 2 a man, is being tracked while another man, ID 3, crosses behind him. Despite the close proximity and potential for confusion, the model successfully maintains the correct IDs for both individuals after they cross paths.

The success in these examples can be attributed to several factors:

- **Feature Extraction:** The tracker extracts deep features from detected objects to create a unique descriptor for each one. This feature extraction is crucial for distinguishing between objects that are visually similar but occupy the same space at different times.

- **Motion Prediction:** The tracker incorporates motion prediction models, such as Kalman filtering, which predict the future position of an object based on its past trajectory. This helps in maintaining consistent IDs during short occlusions or when objects cross paths, as the tracker can predict where each object should be after the occlusion.

- **IoU Matching and Re-Identification:** Intersection over Union (IoU) matching is used to associate detected objects across frames. When objects cross paths, the tracker uses IoU to assess the spatial overlap and re-identifies objects based on their predicted position and appearance features. This mechanism is crucial in maintaining IDs when objects momentarily occupy the same space.

These mechanisms work together to ensure that the tracker can handle complex scenarios where objects cross paths, occlude each other, or move unpredictably, maintaining consistent tracking IDs throughout the sequence.

examples of bad crossing objects that don't keep the same ID:

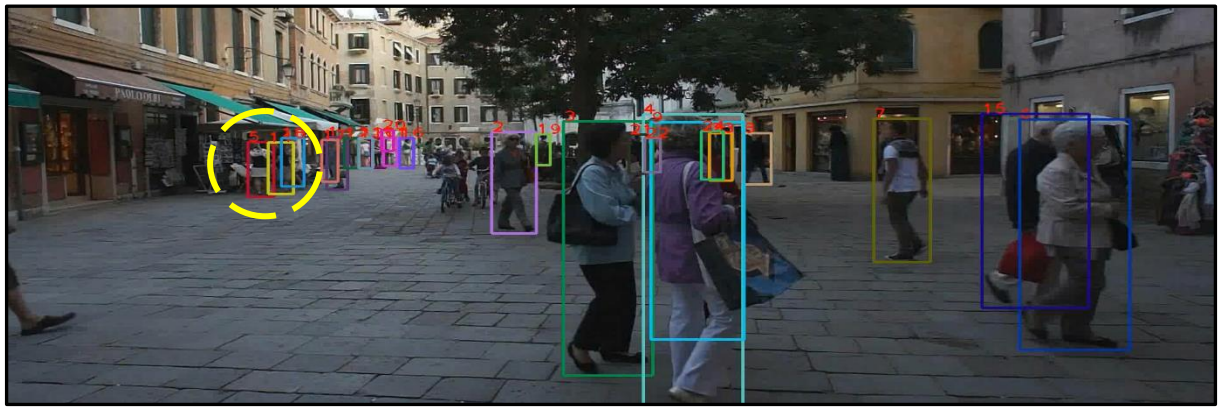


Figure 27 – MOT17-02 frame 1

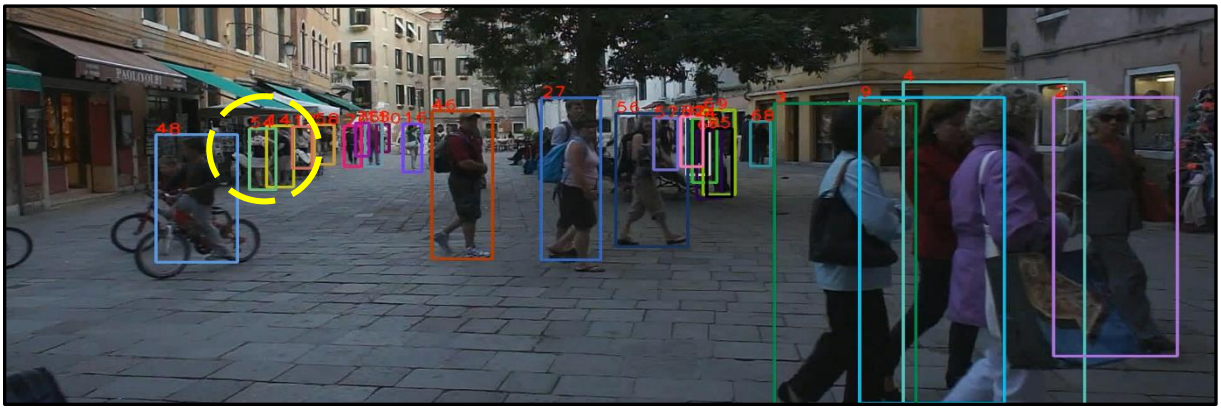


Figure 28 – MOT17-02 frame 199

In this scene, a person with ID 5 (marked with a yellow dashed circle) is being tracked, but as a group of people and a cyclists block the view, the tracker loses track. After about 100 frames, the ID changes to 54, indicating that the tracker was unable to maintain the correct identity through the occlusion.

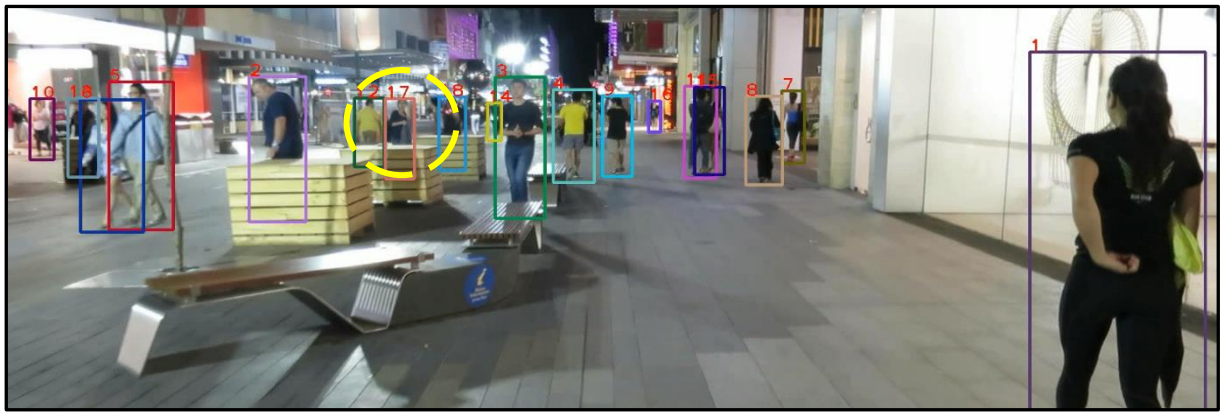


Figure 29 – MOT17-10 frame 4

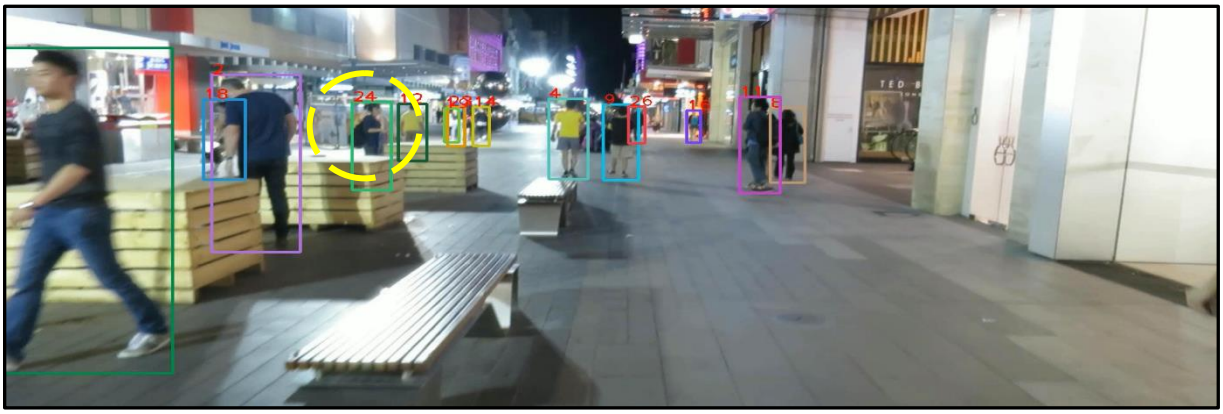


Figure 30 – MOT17-10 frame 79

In this scene, a woman with ID 17 (marked with a yellow dashed circle) is being tracked, but as a group of people block the view, the tracker loses track. The ID changes to 24, indicating that the tracker was unable to maintain the correct identity through the occlusion.

These tracking failures during occlusion are often due to the tracker's inability to effectively handle situations where objects are temporarily blocked or overlap significantly with others. The loss of visual continuity and the difficulty in predicting where and how an object will reappear contribute to these ID switches. To address the issue, one approach could be to enhance the pedestrian descriptors, as the matching between detections and the tracker relies heavily on these features. By creating more distinctive descriptors, the tracker could better maintain the identity of the original target, even during occlusions. Another possible solution is to adjust the balance between false positives and handling occluded cases. Currently, the algorithm tends to switch the tracker to another visible pedestrian when the original target is hidden. By softening this behavior, the tracker could be encouraged to continue tracking the occluded person using the predicted bounding box from the Kalman Filter, rather than switching to a new target.

bad examples of detection that affect on tracking:

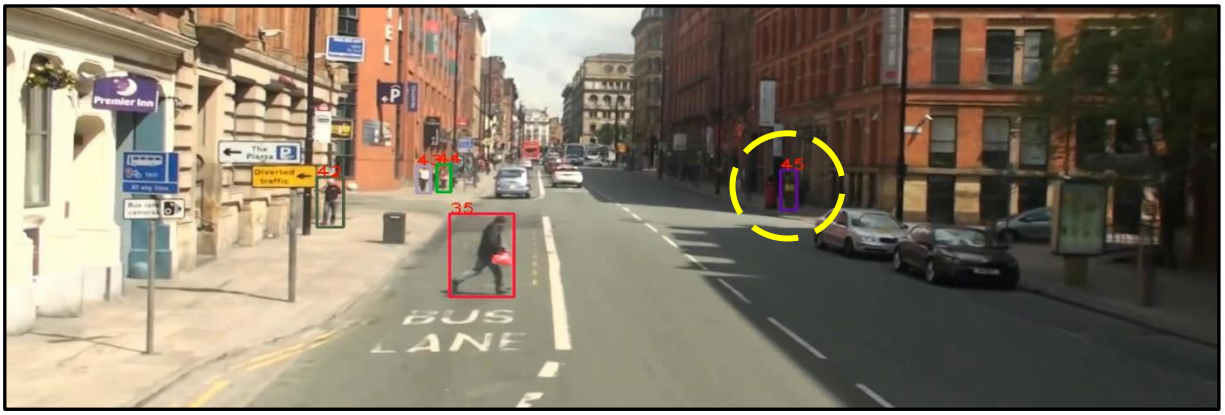


Figure 31 – MOT17-13 frame 316

Figure 32 – MOT17-13 frame 357

In this scene, the object is first detected around frame 316 but then disappears until approximately frame 361. The pedestrian is distant and difficult to spot, even to the naked eye, making this a challenging case. This is an example of a false negative, where the object is present but not detected. As a result, the tracker loses the target for several frames until the algorithm manages to re-detect the pedestrian, leading to a temporary loss of tracking.

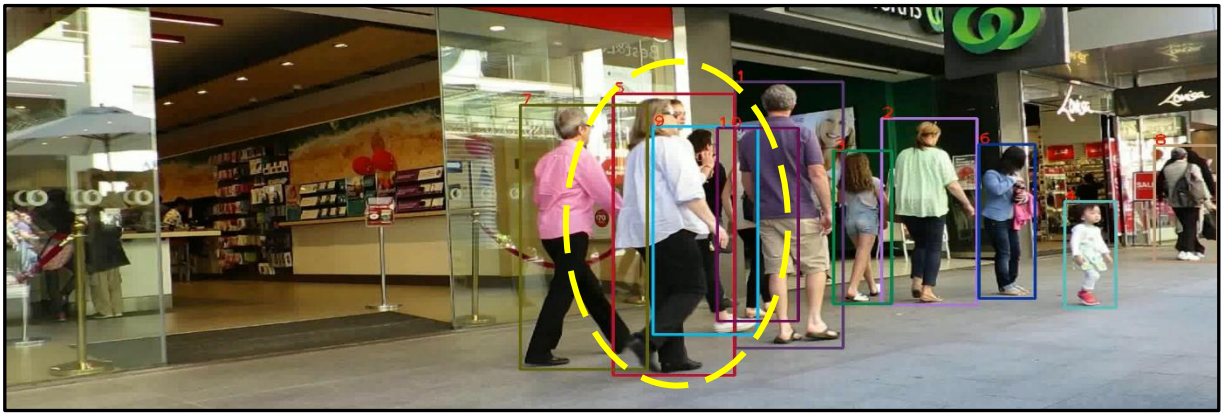


Figure 33 – MOT17-09 frame 1

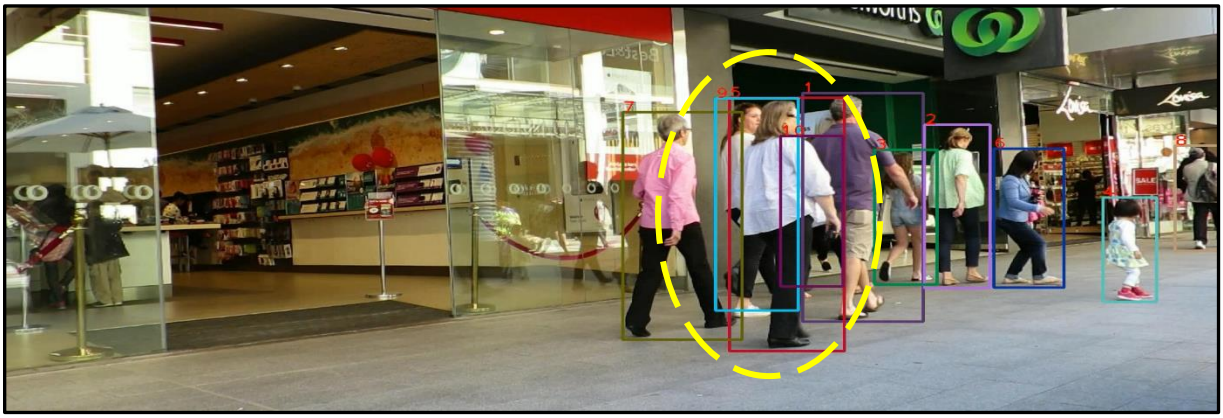


Figure 34 – MOT17-09 frame 16

In the first frame of this scene, the woman with ID 5 partially obscures another woman with white shoes, resulting in a missed detection. This issue persists until around frame 16, when another tracker with ID 9 eventually detects the woman. Since there was no initial detection, the algorithm doesn't recognize that a person is missing from the scene, leading to a complete lack of tracking for the obscured woman during those frames.

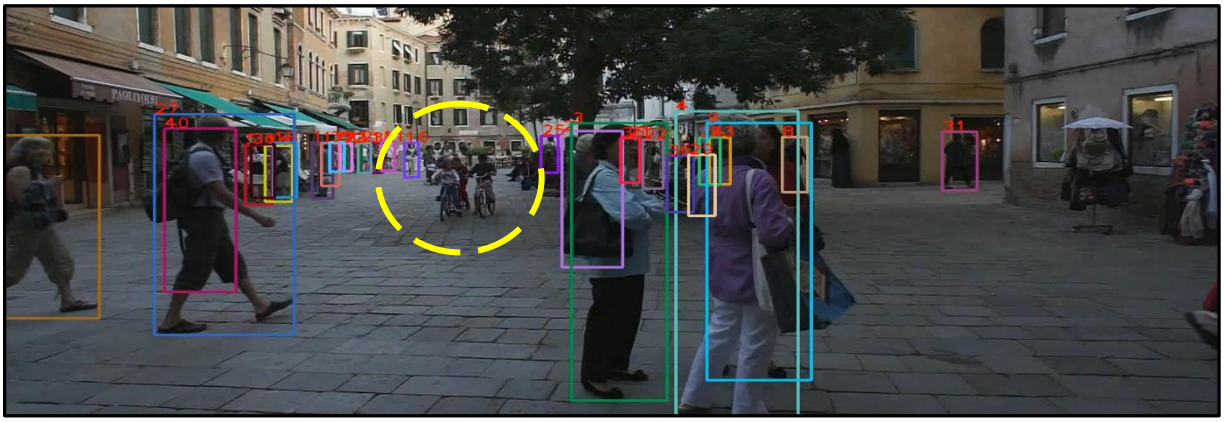


Figure 35 – MOT17-02 frame 40

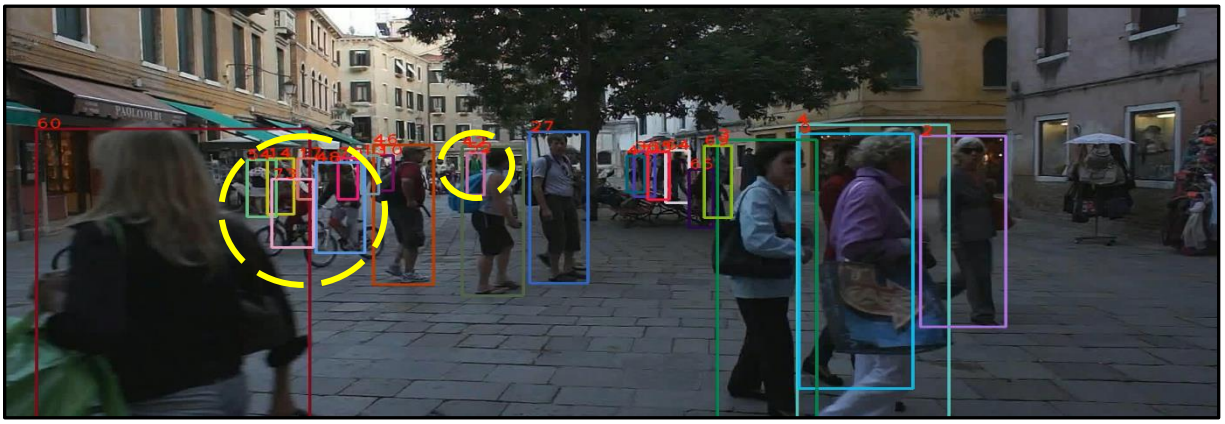


Figure 36 – MOT17-02 frame 169

In this scene, there is a distant group of people and children riding bicycles. The algorithm fails to detect this group in frame 40, likely due to the large distance and the fact that they are moving close

together, making identification challenging. It is only after the group advances and spreads out slightly, around frame 169, that the algorithm begins to detect some of the people and children, but even then, not all are fully identified. This is a case of False Negative, where the objects are present in the scene but are not detected, leading to a loss of tracking for several frames until the algorithm manages to recognize them again.

Appendix

In the Code folder, you'll find two code files: one for Section A of the Visual Odometry, named VisualOdometryEx4, and another for Sections B and C, named Ex4_exercise_final_2024B.

The Figures folder contains all the images included in the report, as well as Table 4.

The Video folder contains the animation video from Section A and the five animations from Section C of the MOT17 sequence.