TEL AVIV UNIVERSITY אוניברסיטת תל-אביב

THE IBY AND ALADAR FLEISCHMAN
FACULTY OF ENGINEERING

הפקולטה להנדסה
על שם איבי ואלדר פליישמן

Project3

Mapping and perception for an autonomous robot/ 0510-7591

By: Tomer Shimshi

Tel Aviv University

August 2023

## Abstract:

This project focuses on three topics: Particle Filter, Iterative Closest Points (ICP), and Visual Odometry. Each topic involves practical implementations and analysis of results.

Particle Filter:

The Particle Filter section introduces the concept of sampling a distribution using particles, generating predictions based on motion models, and updating the predictions using measurements and their uncertainties. The implementation involves loading Odometry and Ground Truth (GT) data, plotting the GT trajectory and landmarks, and utilizing a spin LiDAR 2D sensor for measurement. The Particle Filter algorithm is then executed with the noised GT samples The results are analysed by comparing them to the GT trajectory, calculating Mean Square Error (MSE), and determining the minimal set of particles that maintain good performance. The advantages and disadvantages of Particle Filters are discussed, along with suggestions for overcoming the limitations.

ICP - Iterative Closest Points:

The ICP section focuses on implementing the Iterative Closest Point algorithm on two different scan frames under varying conditions. The vanilla ICP algorithm is implemented using a KDTree for data association. Three different mechanisms and conditions are tested: full point cloud with KDTree associations, filtered point cloud with KDTree associations, and filtered point cloud with Nearest Neighbors associations. The results are analyzed by comparing the point clouds before and after scan correction, examining convergence time, error, and the performance of different association methods. The success and failure of the scan correction process are analyzed, along with potential reasons for mismatched objects.

Visual Odometry:

The Visual Odometry section aims to implement a monocular visual odometry pipeline using essential features such key point tracking, pose estimation, and refining of rotation and translation. The dataset used is the KITTI visual odometry dataset, and the algorithm is calibrated to ensure a maximum Euclidean distance of 15 meters from the ground truth within the first 500 frames. The algorithm pipeline is described, and the results are analysed through animations, comparison with the ground truth trajectory, and identifying reasons for drift.

Overall, this assignment provides hands-on experience with Particle Filters, ICP, and Visual Odometry, along with in-depth analysis and suggestions for improvement. The practical implementations are carried out using a Google Colab notebook for Particle Filter and ICP, and a python script on the KITTI visual odometry dataset for Visual Odometry.

## Contents

Solutions:

## 1. Particle Filter:

Particle filters sample a distribution with a collection of particles, generate a prediction of the distribution by forward predicting each particle using motion model and then compare and update that prediction using a measurement and its uncertainty characteristics.

### A.

In the first section we loaded the attached data file (Odometry and landmarks data) And plot the ground truth trajectory starting from ($x_0$ , $y_0$ , $\theta_0$) based on the following motion model:

$$\begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} = \begin{pmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{pmatrix} + \begin{pmatrix} \delta_{trans}\cos(\theta_{t-1} + \delta_{rot1}) \\ \delta_{trans}\sin(\theta_{t-1} + \delta_{rot1}) \\ \delta_{rot1} + \delta_{rot2} \end{pmatrix}$$

The ground truth we calculated and landmarks are shown in the next figure:

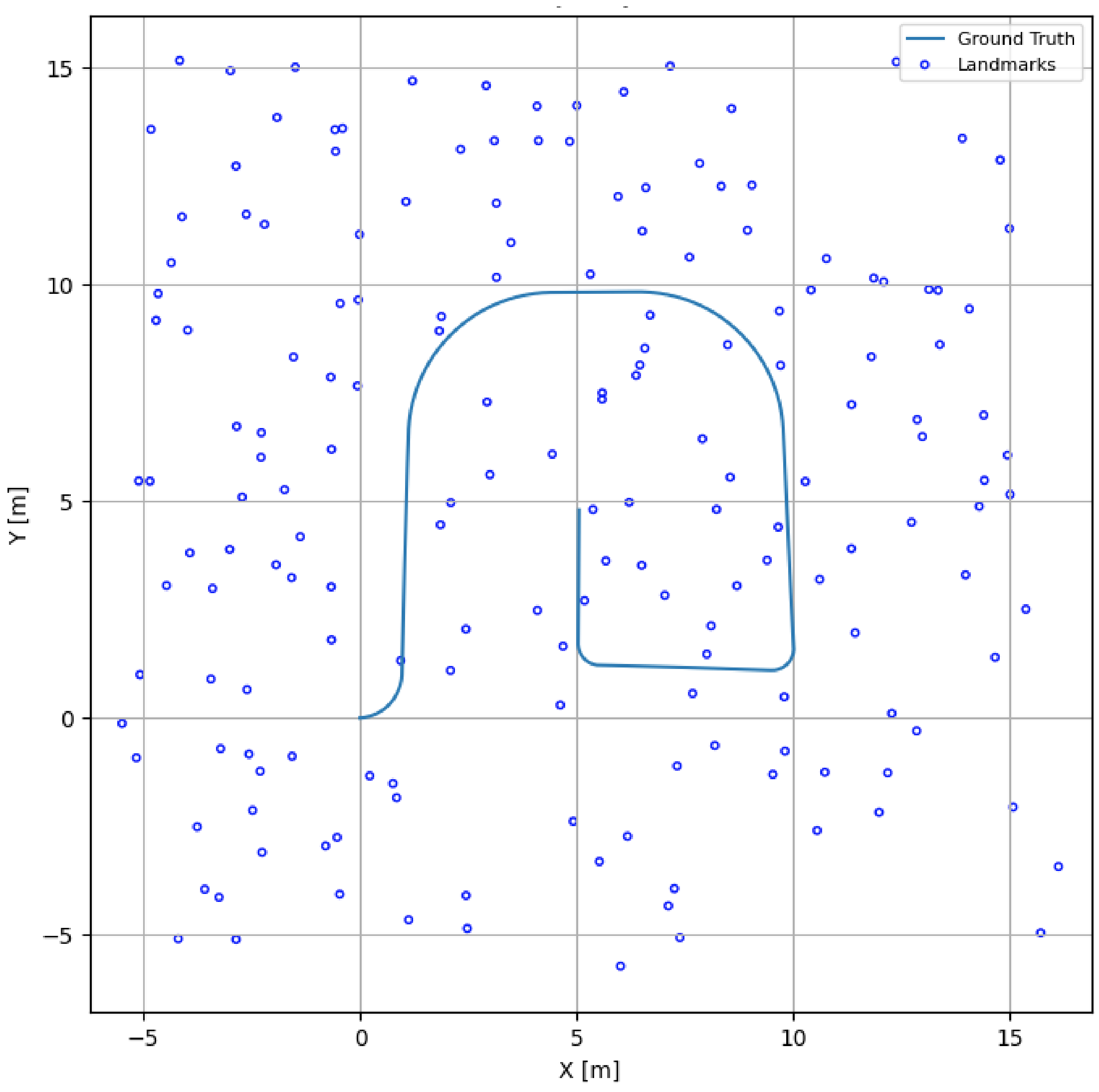


*Figure 1 GT trajectory and landmarks*

We can see that we have an arched path ending at the centre of the area the landmarks are populated throughout the map.

## B.

For the task of testing the Particle Filter we have added noise to the measurement odometry ($\sigma_{r1} = 0.01$ , $\sigma_t = 0.1$ , $\sigma_{r2} = 0.01$) and receive the following noisy trajectory.

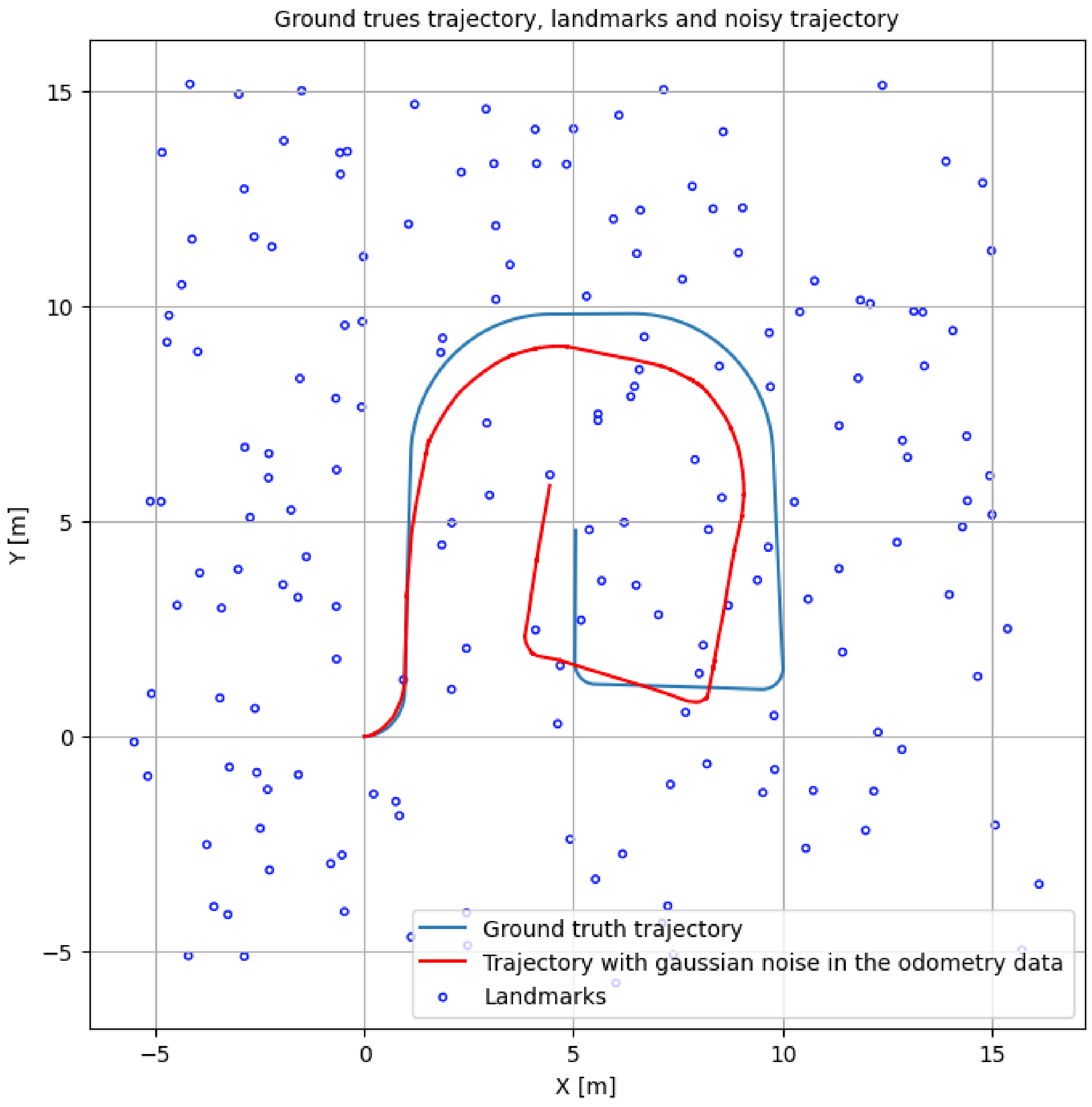


*Figure 2 Ground trues trajectory, landmarks and noisy trajectory*

## C.

In this section we initialized a particle array with N = 1000 particles each containing 6 elements: $pose$ = $(x, y, \theta)$ $weight$ = $w$ $sensor\ measurement\ z$ = $(r,\varphi)$

The pose of each particle was initialized to:

$$x_0 \sim N(0,2), y_0 \sim N(0,2), \emptyset_0 \sim N(0.1,0.1)$$

Such that all particles got a slightly different pose around the assumed starting point.

The weight was initialized to$W = \frac{1}{N}, s.t\ N = number\ of\ particle$

Sensor measurement was calculated after first motion step for each particle.

We then apply a noisy motion model on each particle, making every particle move a little different then the others with the added noise as in section b has standard deviation of: $\sigma_{r1} = 0.01$ , $\sigma_t = 0.1$ , $\sigma_{r2} = 0.01$.
This allows for the model to cover more location possibilities and be more "flexible". all particles move slightly differently such that even if we missed the ground truth location our particles can move towards it.

We then apply our observation and update all particle Sensor measurement giving us $z$ = ($r$,$\varphi$) for each particle.
For each particle we find the closest landmark to it and we then want to use the difference between the ground truth closest landmark and the particles closest landmark.
We then give all the particle's new weights based on their normal Mahalanobis distance from the ground truth measurement. their normal Mahalanobis distance is defined as: A measure of the distance between a point P and a distribution D, meaning It is a multidimensional generalization of the idea of measuring how many standard deviations away P is from the mean of D.
For a normal distribution in any number of dimensions, the probability density of an observation $\vec{x}$is uniquely determined by the Mahalanobis distance d:

$$\Pr[\vec{x}]\, d\vec{x} = \frac{1}{\sqrt{\det(2\pi Q)}} \exp\left(-\frac{(\vec{x}-\vec{u})^T Q^{-1} (\vec{x}-\vec{u})}{2}\right) d\vec{x} = \frac{1}{\sqrt{\det(2\pi Q)}} \exp\left(-{d^2}/{2}\right) d\vec{x}$$

Where in our case Q= $\begin{pmatrix} \sigma_r & 0 \\ 0 & \sigma_\varphi \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0.1 \end{pmatrix}$ is the non-singular covariance matrix.
this way we can compute how
we then normalize the weights by dividing by the total weight of all particles making each weight to be between 0 and 1 and resample according to the normalized weights distribution.
At the end of the processes we choose the particle that had the highest weight in each iteration as the trajectory estimate.

## D Analysis Results:

1. As explained in previous section, the main steps of the Algorithm are as followos:
   Initialize particle's distributed in the robots area
   - For all time steps:
   1. Perform propagation of particles according to motion model
   2. Weighting of particle's according to likelihood of their observation
   3. Resampling
   4. Return to step 1
   - Choose the best particle of each time step (used to show the trajectory)

   We will illustrate the steps using the figures, first we will illustrate the creation of all particles, This is Done with a Particle Filter containing 500 particles, as we have measures that this amount of particles gives the smallest MSE.

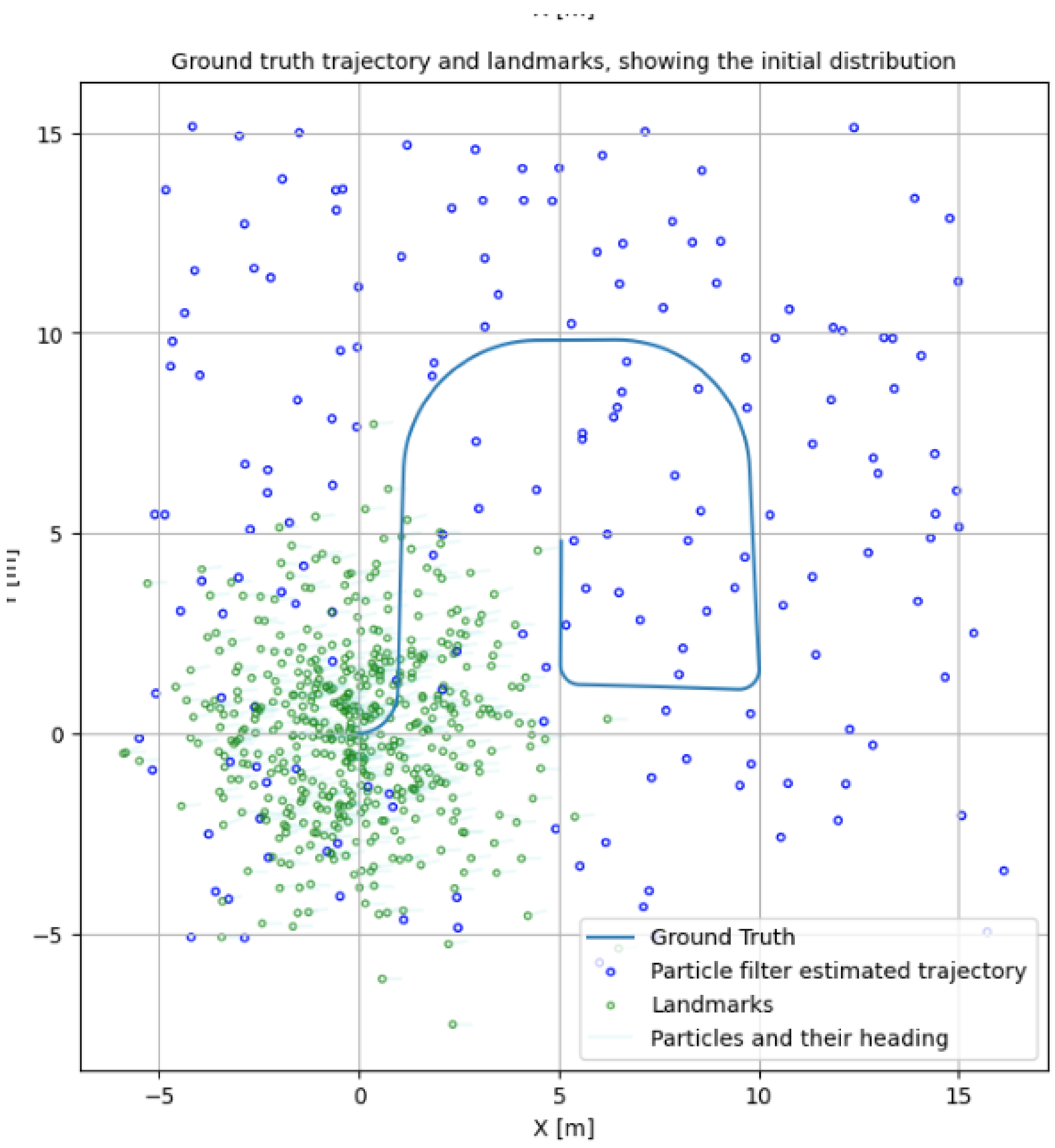


*Figure 3 PF initialization*

In the image bellow we can see the particles and their heading in the first frame

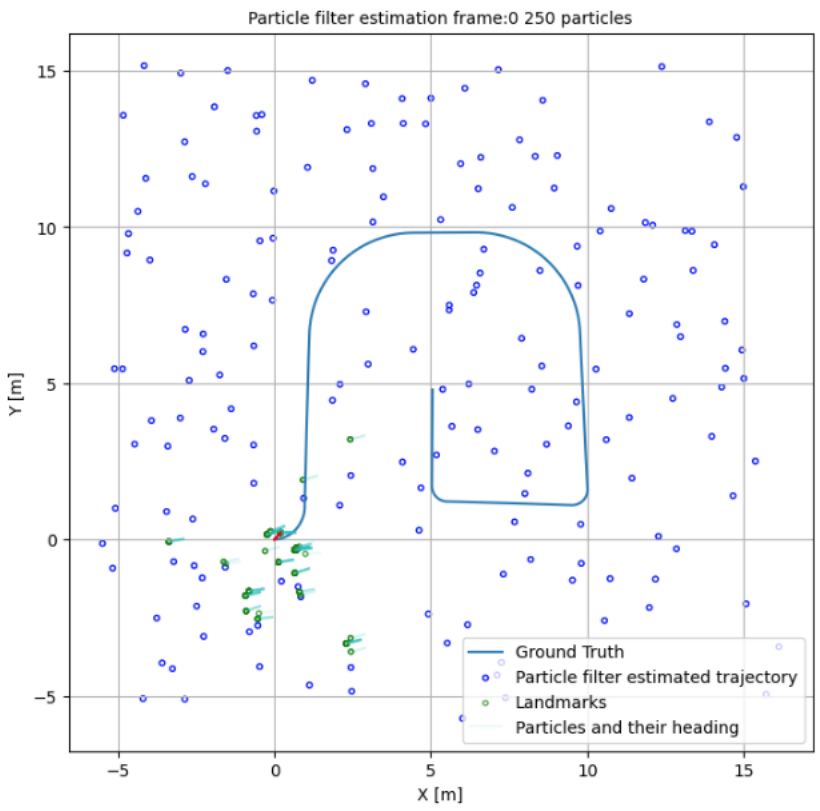


*Figure 4 PF estimation frame 0*

As we can we have a lot of particles heading different locations since it is just the first frame.

After a few iterations we can see that the PF manged to get a descent approximation of the ground truth trajectory.

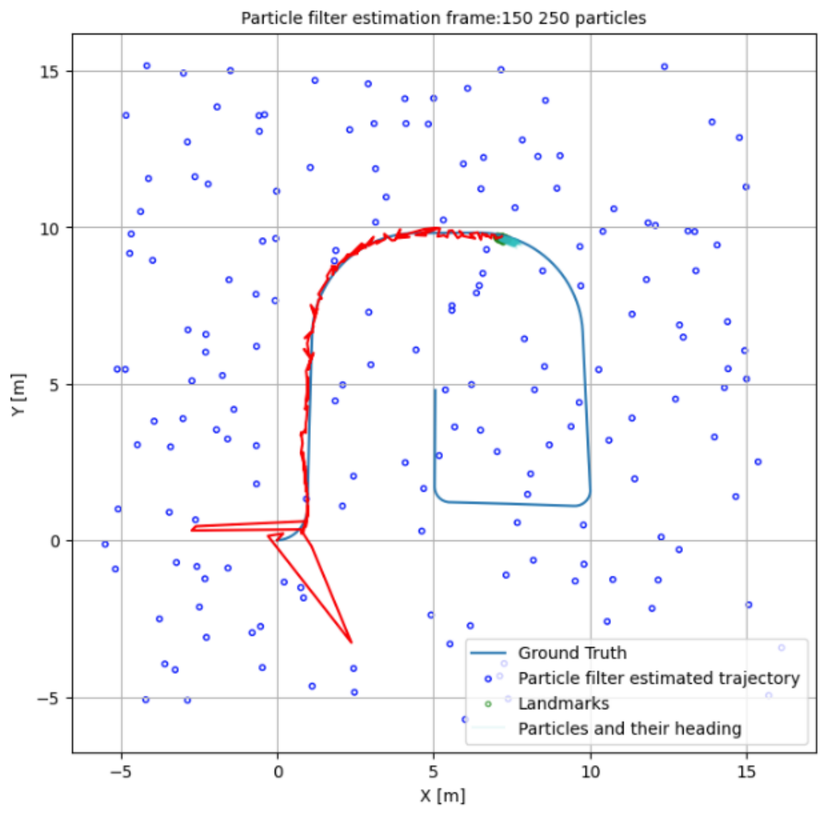


*Figure 5 PF estimation frame 150*

As can be seen the particle's are slowly converging to the true location of the robot .

particles that have similar measurement to the true measurement survive. However, we

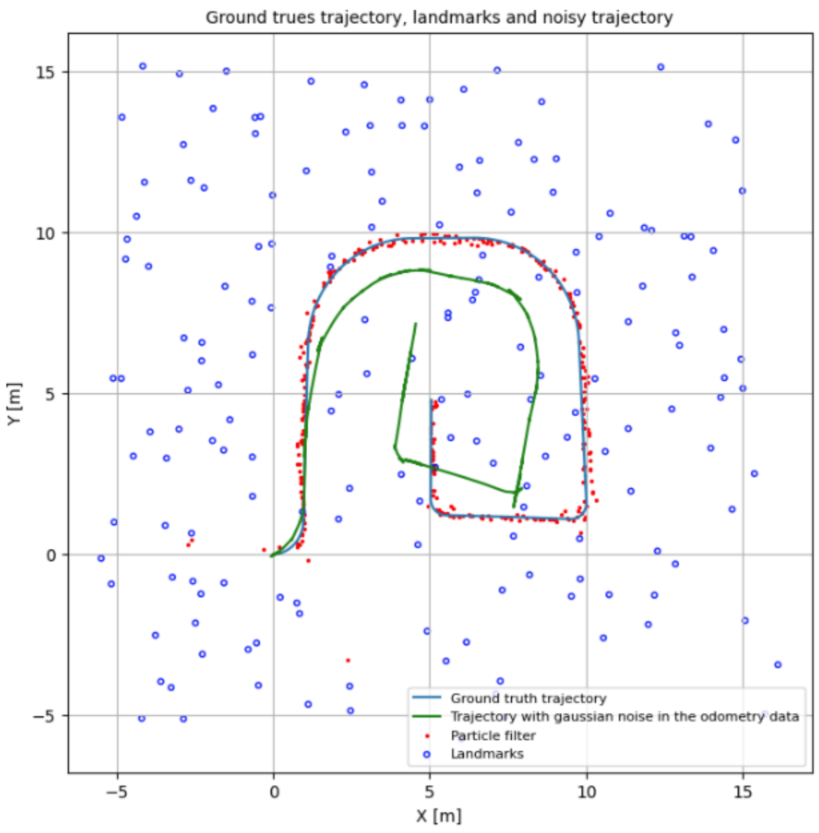
already see the problem in the way of our observation, because we only take the

measurement from the closest land mark there are a lot of states the particle's could be

in that would still be similar to the ground truth location but are not the true location we

will conclude later that using more landmark observations will make the ground truth

pose more unique and hence the estimation more exact, ie less particles will have hi



probability.

*Figure 6 trajectory comparison*

As we can we got a good trajectory estimation, on top of that we can see (also in the attached mp4 videos) that it took the filter roughly 10 frames until is really started fitting itself to the data, we can also see that in the sharp turns the PF had a bit of a hard time fitting itself to the true trajectory as expected but overall we can say that the PF managed to mimic the true trajectory even with the noisy datasampels.

We have also calculated the MSE ranging from the 10$^{th}$ frame

MSE = 0.06 m

2. We can see that the PF managed to find a trajectory that is very near the ground truth, this is due to the fact that PF doesn't really on the gaussian noise model and can fit itself to noise with unknown distribution.
3. We have calculated the MSE of different number of particles to find the minimum number of particles that still maintain good performance

| #particals | 50 | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | 1000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| MSE | 29.4 | 0.045 | 0.043 | 0.06 | 0.106 | 0.09 | 0.066 | 0.077 | 0.1 | 0.091 | 1.6 |

We can see that overall every PF with more than 50 particles can get a good a result, we believe that the reason we got such a big error for 50 particles is due to the fact that we have a lot of landmarks and if we don't have enough particles the PF might fit itself with the wrong landmarks which caused the big MSE. Overall we can see that the quality of the estimation is really dependent on the initial distribution of the particles and the amount of landmarks, on top of that we believe that if we had used more than just the nearest landmark to locate the best particles then we could have gotten better results, for example using the 3 nearest landmarks will give us a better accuracy but with a computational increase. Attached below is the graph comparing the MSE VS the number of particals. Overall we believe that We managed to implement the particle filter in a good way and get results close to the ground truth trajectory.

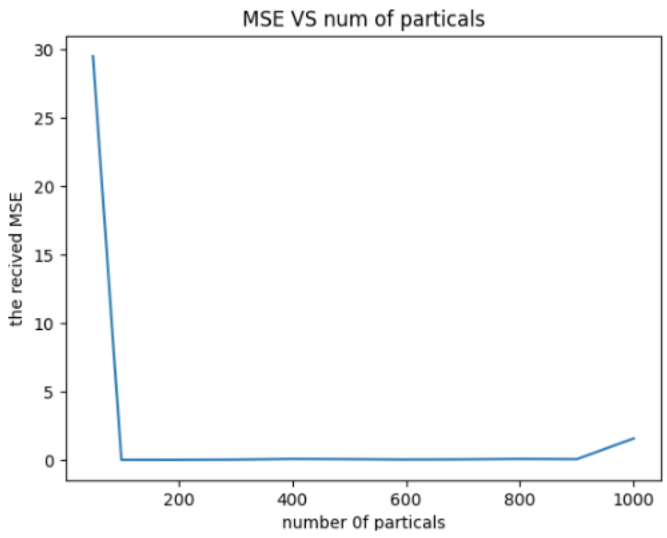


*Figure 7 MSE VS num of particles*

4. **Pros of Particle Filters**:
   Flexibility: Particle filters are highly flexible and can handle non-linear and non-Gaussian distributions. They can effectively represent complex, multimodal posterior distributions, making them suitable for a wide range of applications.
   Robustness: Particle filters can handle uncertainties and noise in both motion models and sensor measurements. They are capable of adaptively adjusting the number and distribution of particles to track the state estimation accurately even in the presence of outliers or abrupt changes in the environment.
   **Cons of Particle Filters**:
   Computational Complexity: Particle filters can be computationally expensive, especially when dealing with a large number of particles or high-dimensional state spaces. As the number of particles increases, the algorithm's time and memory requirements grow significantly, which can limit real-time applications.
   Particle Degeneracy: In particle filters, particles with low weights contribute less to the estimation process, leading to particle degeneracy. Over time, a few particles dominate the

distribution, reducing the diversity and accuracy of the estimation. This issue becomes more significant when the system exhibits high nonlinearity or strong measurement noise.
**Suggestions to overcome the cons and develop a better mechanism:**
Sampling and Resampling Techniques: To mitigate the computational complexity, advanced sampling and resampling techniques can be employed. Techniques such as Sequential Importance Resampling (SIR), Regularized Particle Filters, or Gaussian Mixture Models can improve the efficiency and accuracy of particle filters by adaptively adjusting the number of particles or approximating the posterior distribution.
Adaptive Particle Representation: Introducing adaptive mechanisms to dynamically adjust the particle distribution can help address particle degeneracy. Techniques like Sequential Monte Carlo Methods or Particle Filters with Multiple Importance Sampling can be utilized to maintain diversity among particles and prevent the loss of valuable information. Adaptive mechanisms can consider the importance of different regions of the state space and allocate particles accordingly.
By incorporating these suggestions, researchers can develop enhanced versions of particle filters that exhibit improved computational efficiency, accuracy, and robustness in various applications.

## 2.ICP – Iterative Closest Points

### A.

We have implemented the Vanilla ICP algorithm and completed the code in in the Google Colab notebook (fill #TODO). We will now describe the process:

1. We have loaded the 2 point clouds frame and the corresponding input images as can be seen in the image bellow

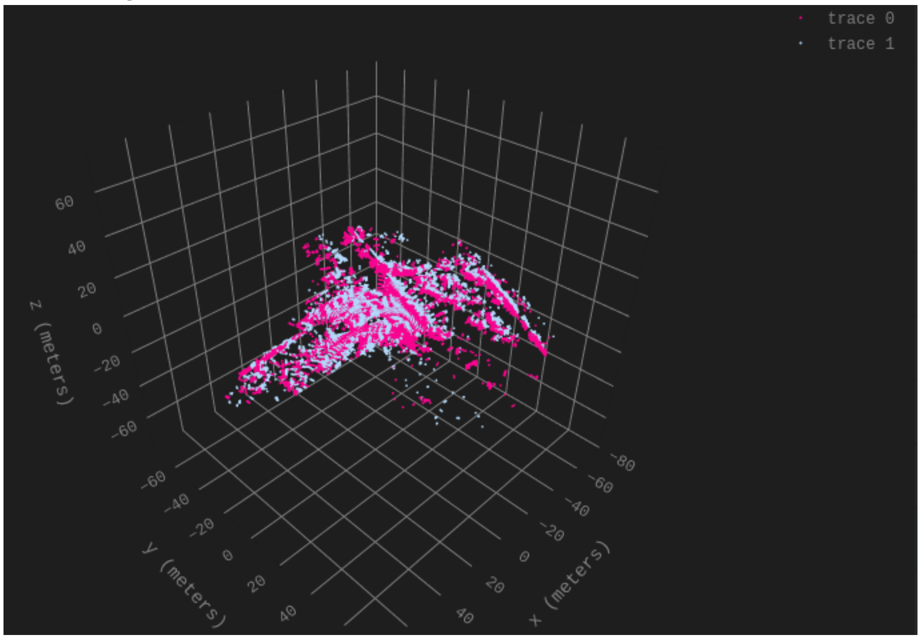


*Figure 8 the 2 point cloud prior to the ICP*

As we can see there exist a divergence between the two point clouds which is caused by the vehicle's movment

2. We implemented the K-D algorithm which allows us a useful multi-dimensional search. K-D Tree holds a space-partitioning data structure for organizing the points in a k-dimensional space. in order to find the indices of point cloud B which are closest to point cloud A
3. We have estimated the transformation via SVD (Singular Value Decomposition) as learned in class.
4. We applied the ICP routine as follows:
    - find the corresponding pairs between the 2-point clouds.
    - estimated the needed affine transform.
    - applied the transform on the point cloud B
    - updated total transformation
    - and run until convergence or until maximum iterations is reached
5. For each point $p_i$ in the source cloud P, find its closest point $q_j$ in the target cloud Q using the K-D algorithm which allows us a useful multi-dimensional search. K-D Tree holds a space-partitioning data structure for organizing the points in a k-dimensional space. to find the indices of point cloud B which are closest to point cloud A.

Compute the transformation:

Compute the centroid (mean) of the source cloud: $\bar{p} = \frac{1}{N} \cdot \sum_1^N p_i$

Compute the centroid (mean) of the target cloud: $\bar{q} = \frac{1}{N} \cdot \sum_1^N q_i$

Compute the covariance matrix W: W = $\sum_1^N \ (p_i - \bar{p}\,) \cdot (q_i - \bar{q}\,)$.

Perform singular value decomposition (SVD) on W: $USV^T$ = SVD(W).

Compute the rotation matrix R: R = $VU^T$.

Compute the translation vector t: $t = \bar{p} - R\bar{q}$

Update the transformation.

Apply the computed transformation to the source cloud: P' = R * P + t.

Check for convergence:

Compute the mean square error (MSE) between corresponding points in the target cloud and transformed source cloud.

If the change in RMSE from the previous iteration is below the convergence threshold (|MSE_new - MSE_old| < epsilon) , exit the loop.

Increment the iteration count.

In summary, the ICP algorithm iteratively finds correspondences between the source and target point clouds, computes the transformation that aligns the two clouds, updates the transformation, and checks for convergence based on the MSE. This process continues until convergence or reaching the maximum number of iterations.

## B

After the process described above, we implement the Vanilla ICP algorithm on the first 2 point clouds, first we will show the BEV image of the point clouds to get a better understanding of the task at hand:

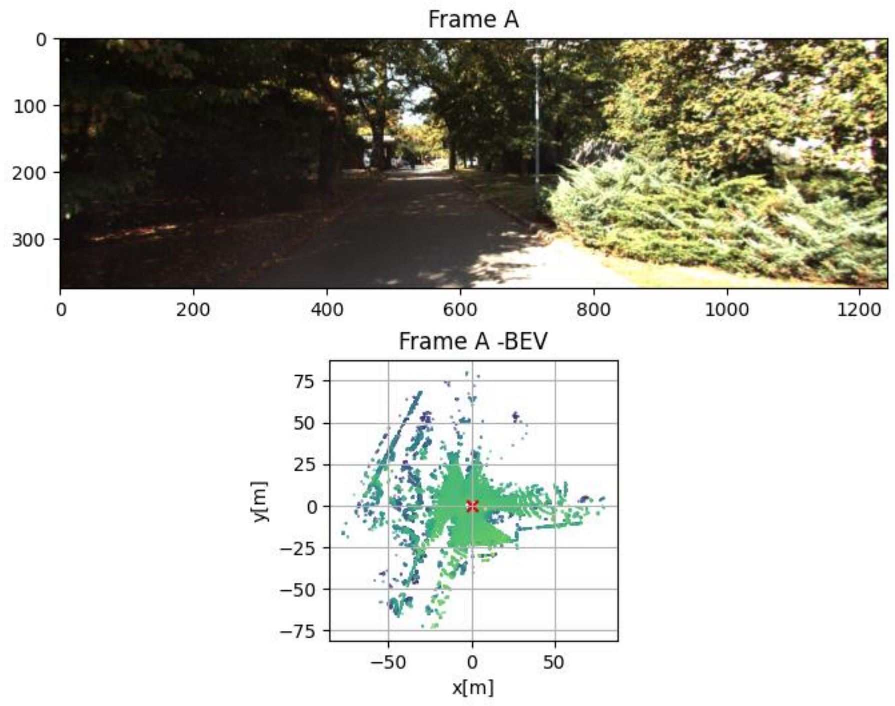


*Figure 9 1st PC frame and BEV*

As can be seen in the image in this case the frame is not a big open area so we would expect the vanilla IPC to perform well even without the clipping of the points belonging to road.

The received animation is added to the finale report submission, and we can see in the below graph how the RMSE drops after each iteration until achieving convergence (as we expected)

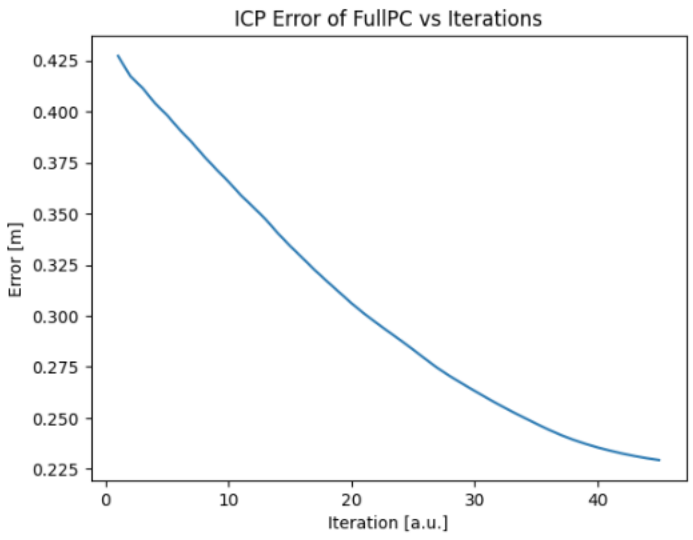


*Figure 10 ICP Error of FullPC VS Iterations*

*lIPC vs iterations*

As we can see the algorithm converged after performing 45 iterations to an error of 0.229 $[m]$.
In the image below we can see the outcome of the algorithm after applying the ICP

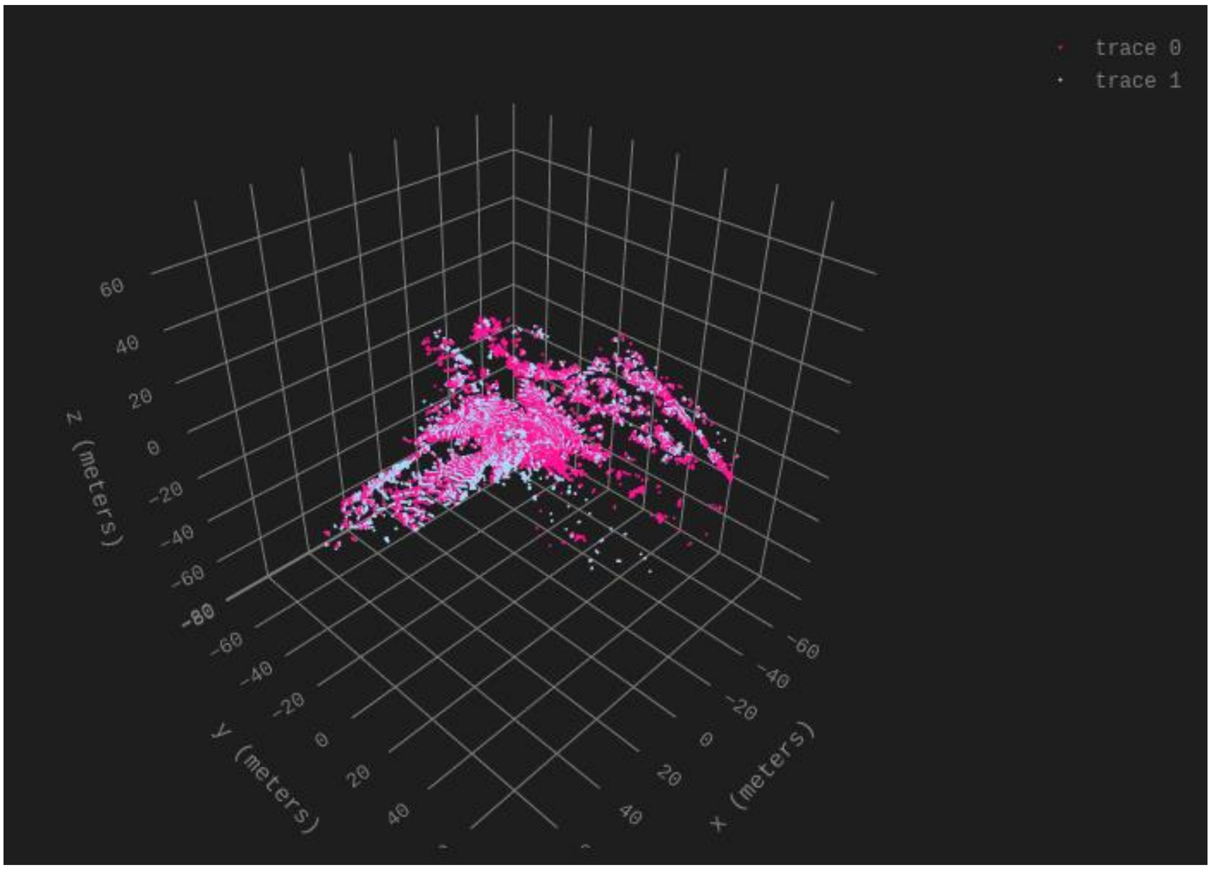


*Figure 11 ICP results*

As we can see, the received point clouds have a smaller divergence between them.

The next step we Run vanilla ICP on filtered point cloud which means that we remove every point in the clouds that is below a certain height (Z value) namely we remove every point that is more than 1 meter bellow the LiDAR height, by doing this we can ignore irrelevant points that are considered as part of the road (כביש) and the point clouds we got are as follows:

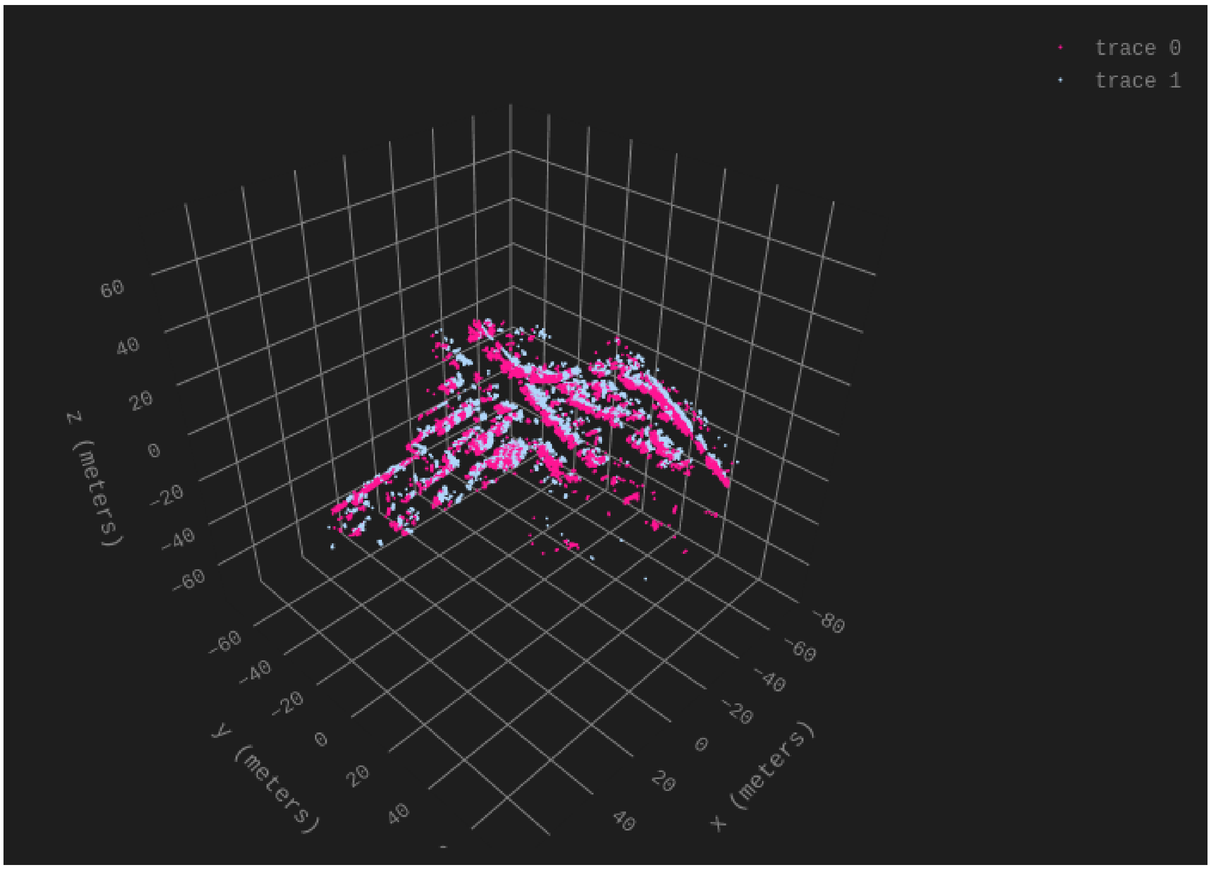


*Figure 12 Filtered PC before ICP*

As we expected the resulting point clouds do not contain parts of the road with them.

Now we can again apply the ICP algorithm on the clipped point clouds and results are as follows:

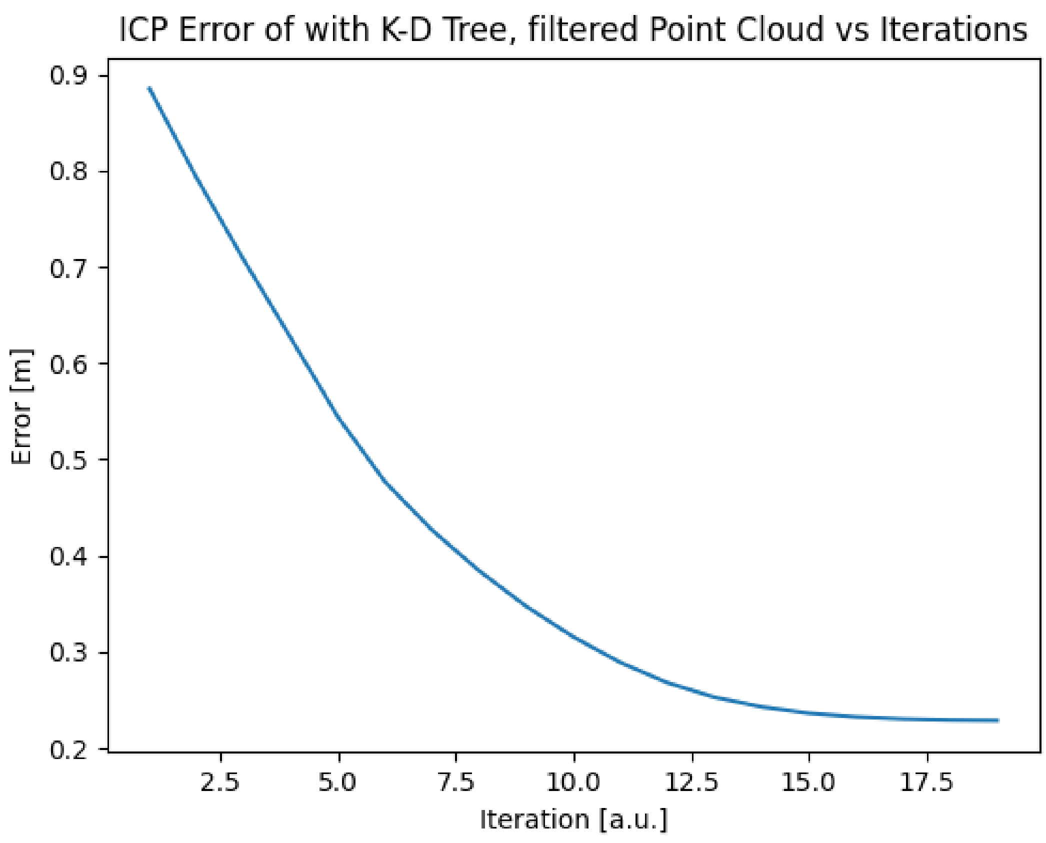


*Figure 13 ICP Error with K-D Tree, filtered Point Cloud*

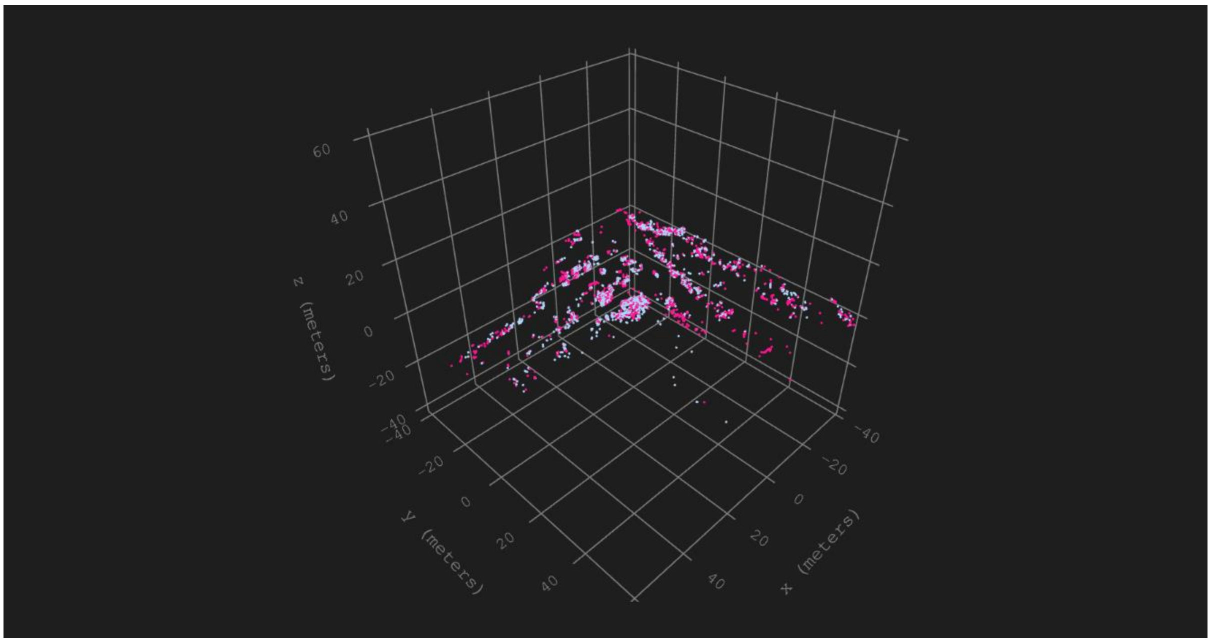


*Figure 14 Filter_PC_ICP-Results*

As we can see the algorithm converged in fewer iterations which makes sense because we now compare less points, but each point is more unique (i.e not part of the road) on top of that we can see the 2 point clouds are very correlated after applying the affine transformation received from the ICP transformation, The entire process can be viewed in the attached HTML file. In terms of the error we can see that the 2 methods converged to the same error which is 0.229[m]

Now we will test the performance of the Nearest Neighbors algorithm on the filtered point cloud, we just applied the same KDTree function, only this time we used a leaf size of 1 instead using the default leaf size which is 40, The leaf size parameter in the KD-tree search algorithm determines the minimum number of points that should be contained within a leaf node of the tree. The choice of leaf size can have an impact on the efficiency and accuracy of the KD-tree search. Larger leaf sizes result in fewer total nodes in the tree, leading to a more compact tree structure. With larger leaf sizes, the tree traversal process can be more efficient since each node contains multiple points, reducing the number of traversals needed to find the nearest neighbors. However, larger leaf sizes can also lead to reduced accuracy in finding the exact nearest neighbors since the points within a leaf node may have larger variations in distance to the query point, With smaller leaf sizes, the tree traversal process can be less efficient due to the increased number of nodes and traversals required. However, smaller leaf sizes typically lead to higher accuracy in finding the exact nearest neighbors since each leaf node contains only one point, allowing for more precise distance calculations..
The received results are as follows:

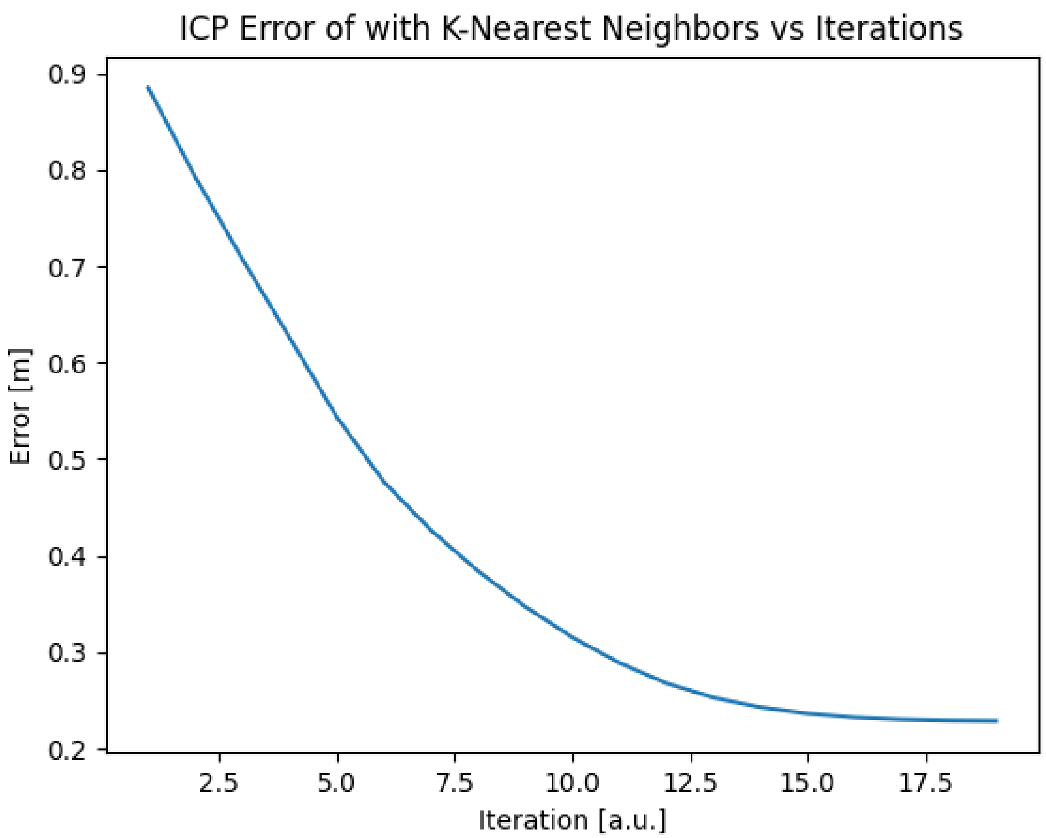


*Figure 15 ICP with K-Nearest Neighbors*

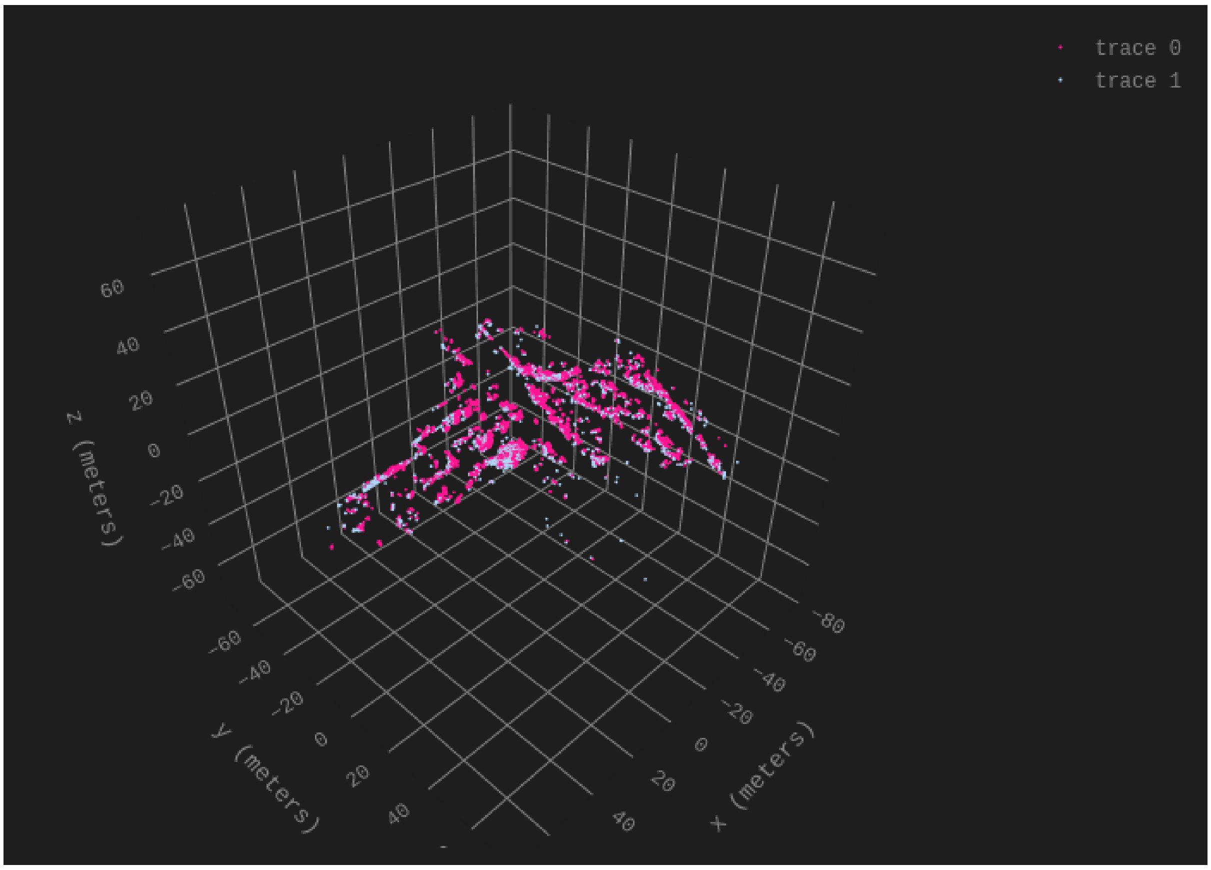


*Figure 16 FilterPC-NearestNeighbors*

As we can see the algorithm managed to converge, rather good, after 19 iteration to an error of 0.229 [$m^2$] as did the others, meaning that in this specific case reducing the leaf size did not resulted in reducing the error but as we can see bellow did resulted in a higher runtime . We can also see that in terms of Elapsed time and finale errors we received the following numbers:

Final error of K-D Tree is: 0.22941560030342173 [m]

Final error of K-D Tree with filtered PC is: 0.22859785615689782 [m]

Final error of K-Nearest Neighbors with filtered PC is: 0.22859785615689782 [m]

Elapsed time kdree: 3.1027 seconds

Elapsed time knn: 4.1707 seconds

Meaning that they all converged to roughly the same distance value but we can clearly see that the kdtree took less time to converge

In the graph below we can see the comparison of error VS iteration of each algorithm

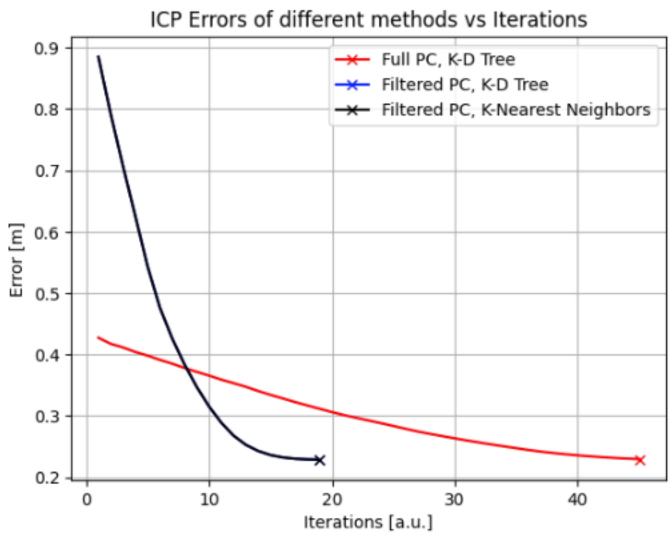
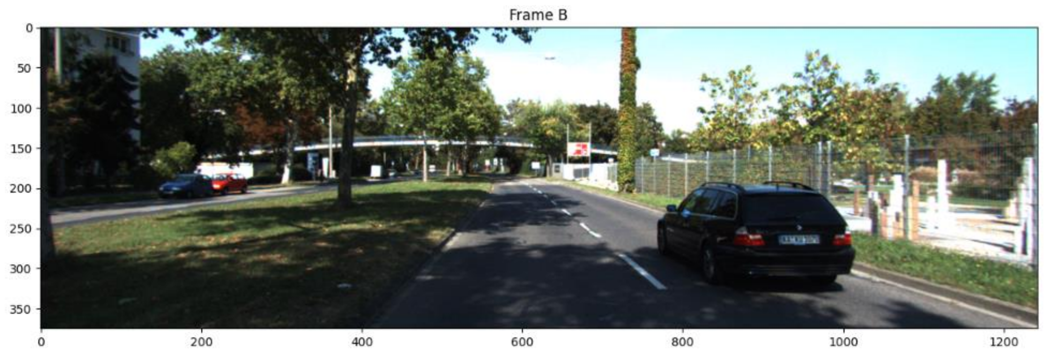


*Figure 17 ICP Errors of different methods vs iterations*

As we can see the results of the filtered PC K-D tree and KNN overlap with one another, and since the point clouds start with less points (i.e filtered) it starts with a bigger error but converge faster.

## C

Now we will apply the same analysis as we did in section B only this time, we will use record 2011_09_26/0013. Use Point clouds 10 and 15, first we will again show the BEV and image of the point cloud to get a better understanding of the task at hand:



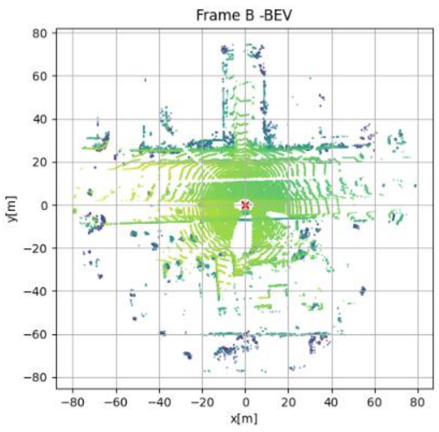


*Figure 18 2nd PC frame and BEV*

As we can see this time the frame is in a less constrained space (more open) meaning we will have more parts of the road which as we learned in class, is a problematic surrounding for the vanilla IPC on the full PC.

For starters we will at the 2 new point clouds without applying any change to them

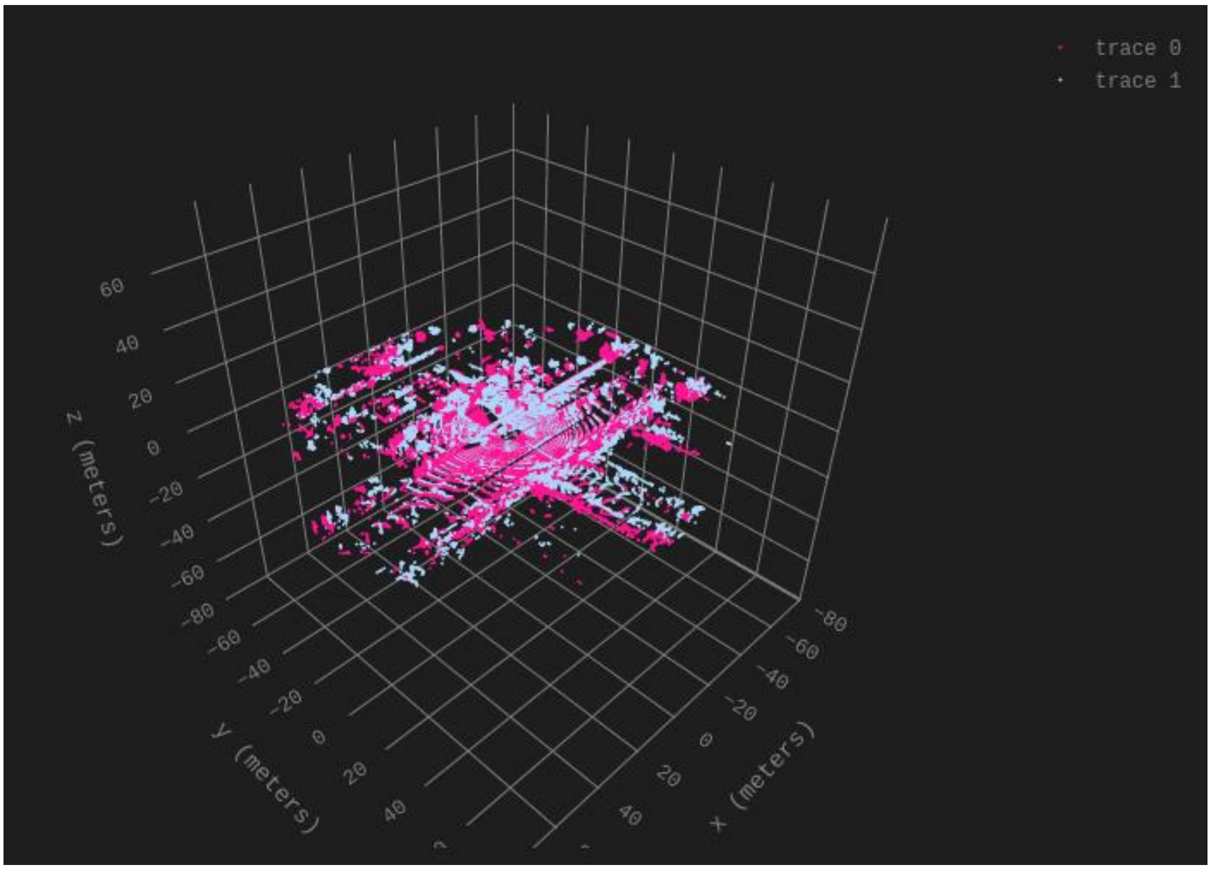


*Figure 19 full PC2 before IPC*

As we can see there id a shift between the 2 point clouds, and this time the soundings area is less bounded than the are of the point clouds we previously checked.

Now we will show the received results after running the ICP algorithm on the full PC

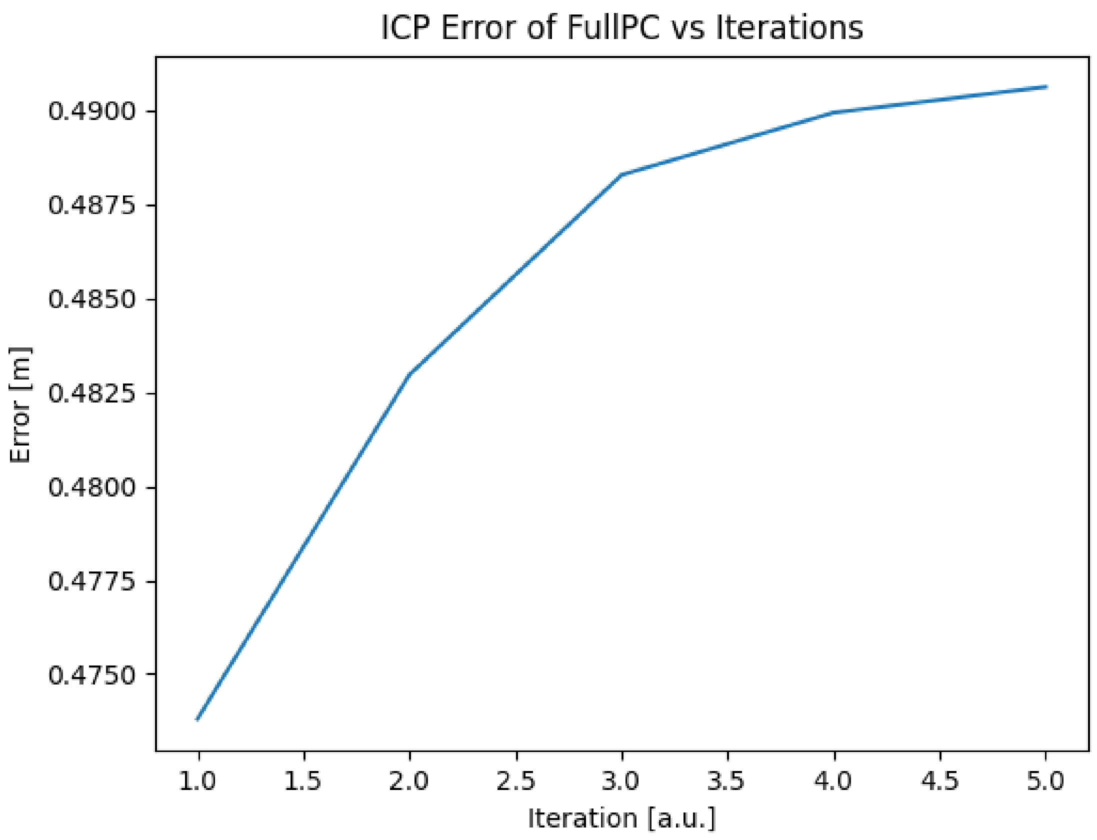


*Figure 20 ICP Error full PC_P2*

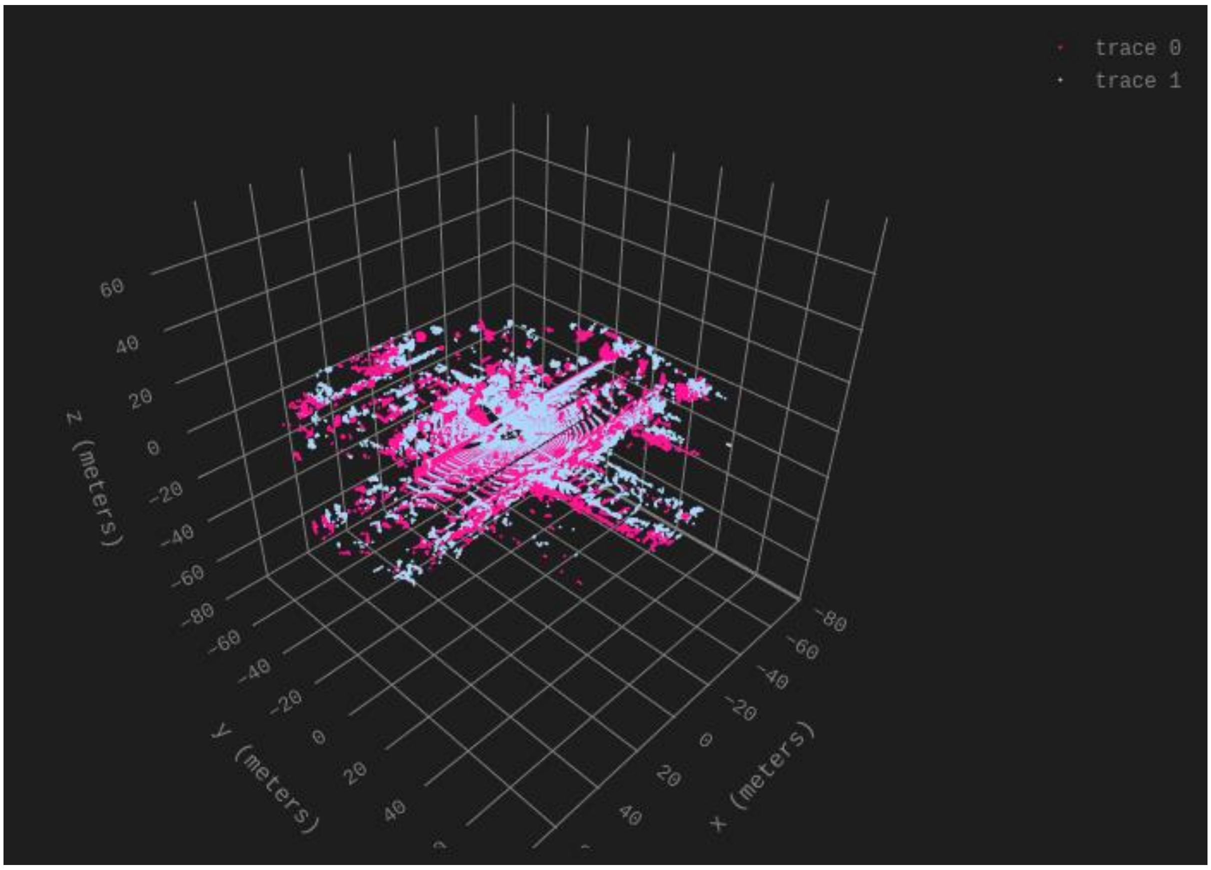


*Figure 21 ICP results P2*

As we can see this time the error increased instead of decreasing after each iteration. We believe that is caused by the usage of the full PC meaning with the road parts  which led the K-D-Tree to compare points that belong to the road with one another, as explained above we can see that in this frame there are more points that belong to the road due to fact that this frame is in a more open area. We can see that the algorithm converged after 5 iterations. Looking at the video itself (which is added to this report in the Results folder, named as FullPC_P2.html) we can really see how applying the ICP actually worsen the correlation between the 2-point clouds)

Now we will apply the same only this time on filters point clouds, first we will show the received PC after the filter step:

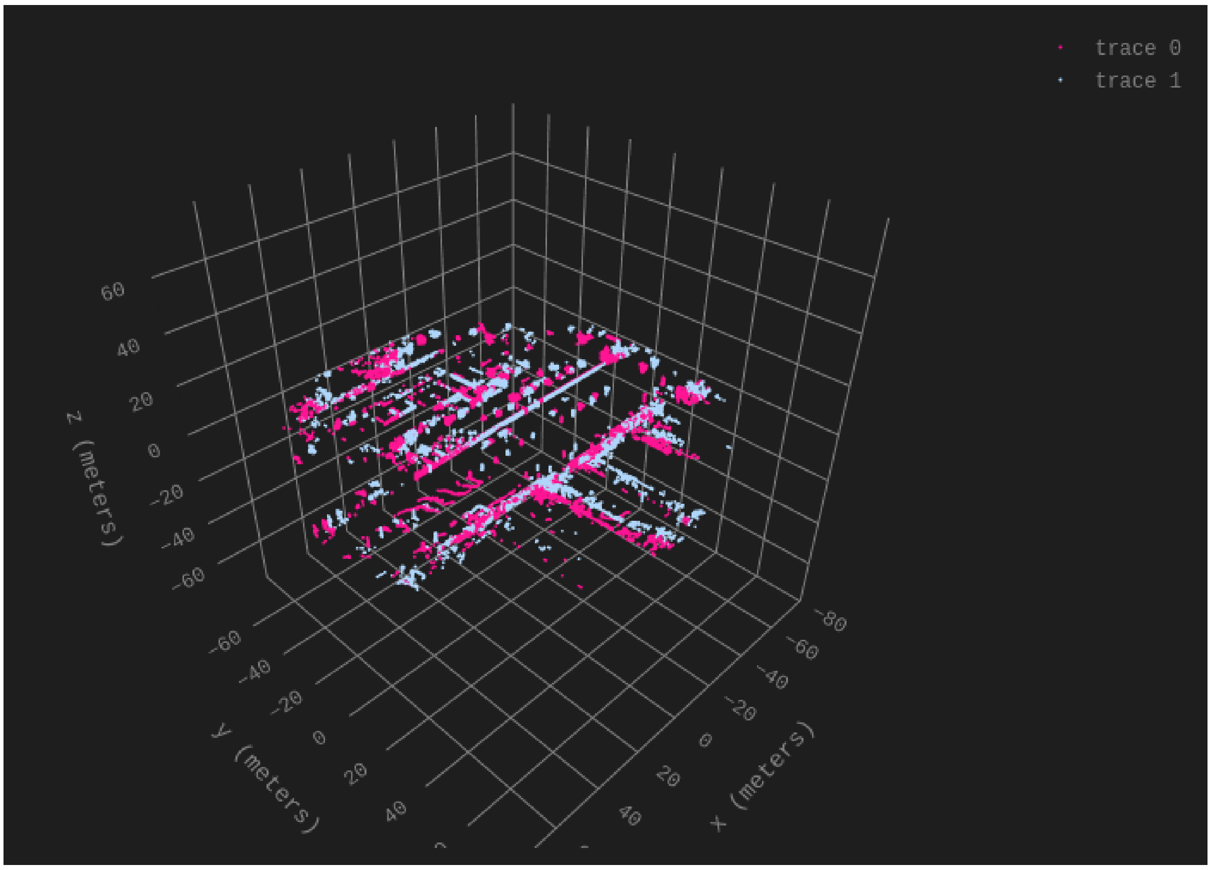


*Figure 22 FilteredPC- before_icp_P2*

As we can see due to the openness of the surrounding area the amount of point is reduced quit drastically, we believe that by applying this pre processing step we will get better results after running the ICP.

We will now look at the received output of the ICP algorithm:

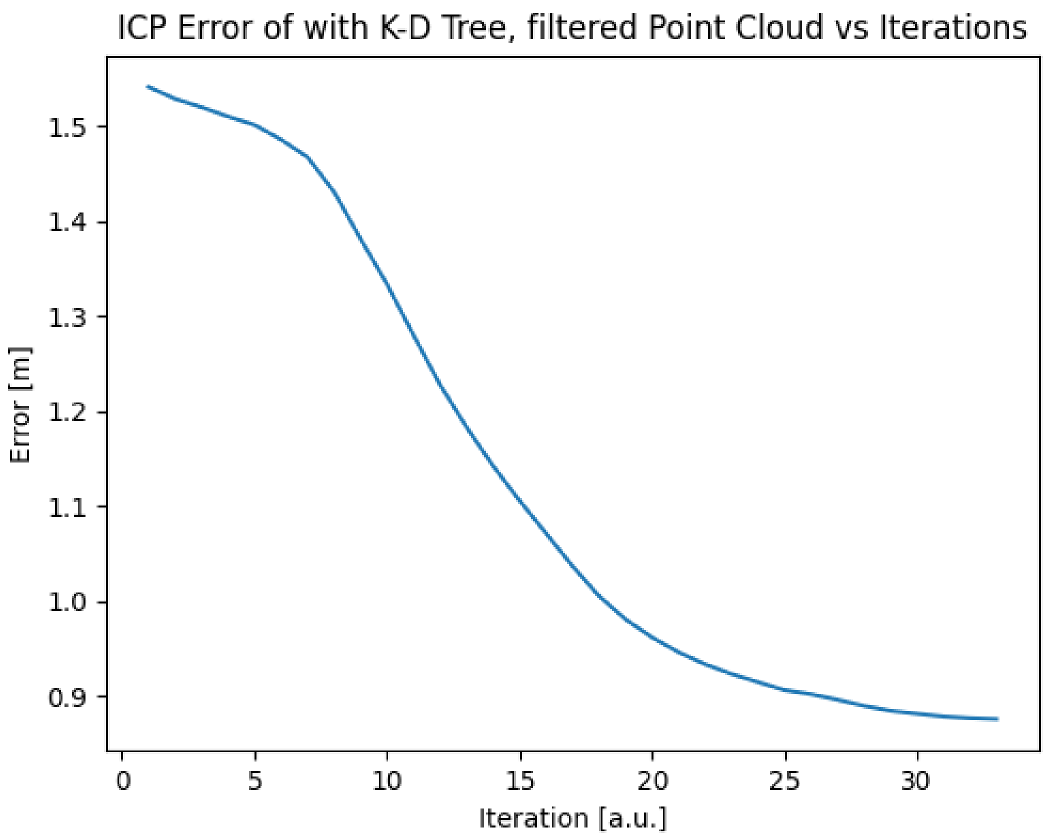


*Figure 23 ICP with K-D Tree, filtered Point Cloud_P2*

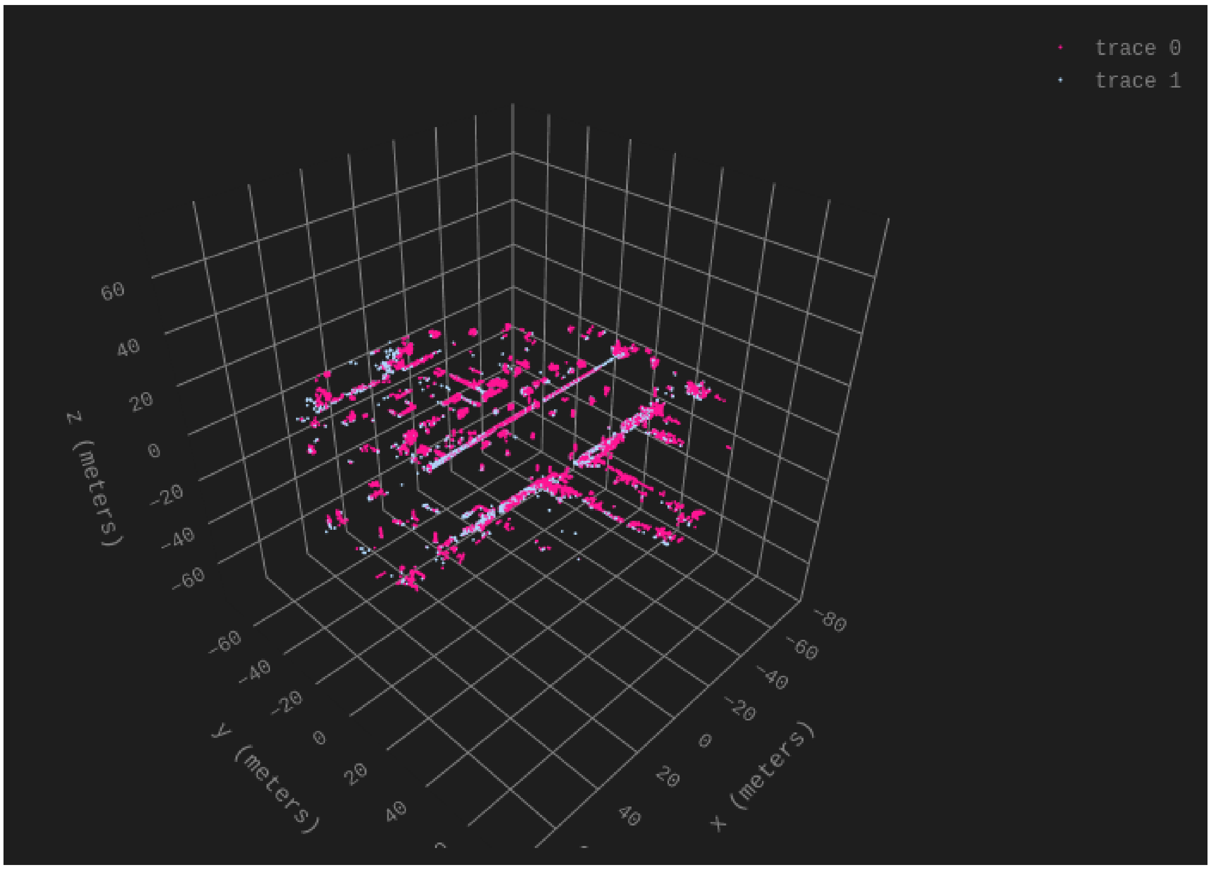


*Figure 24 Filter_PC_ICP-Results_P2*

As we can see after removing the point which corresponds to the road we get a very good correlation between the 2 PC, we can also see it very nicely in the attached html video. We can see that algorithm now managed to converge after 33 iterations which is more than what it took for the previous frame, we believe that this is due to the bigger divergence between this frame PC compared to the frames tested on the previous section.

Now we will apply the KNN on the filtered PC and compare the results, as explained in the previous section for this algorithm we would expect to see a decrease in the error and an increase in the runtime:

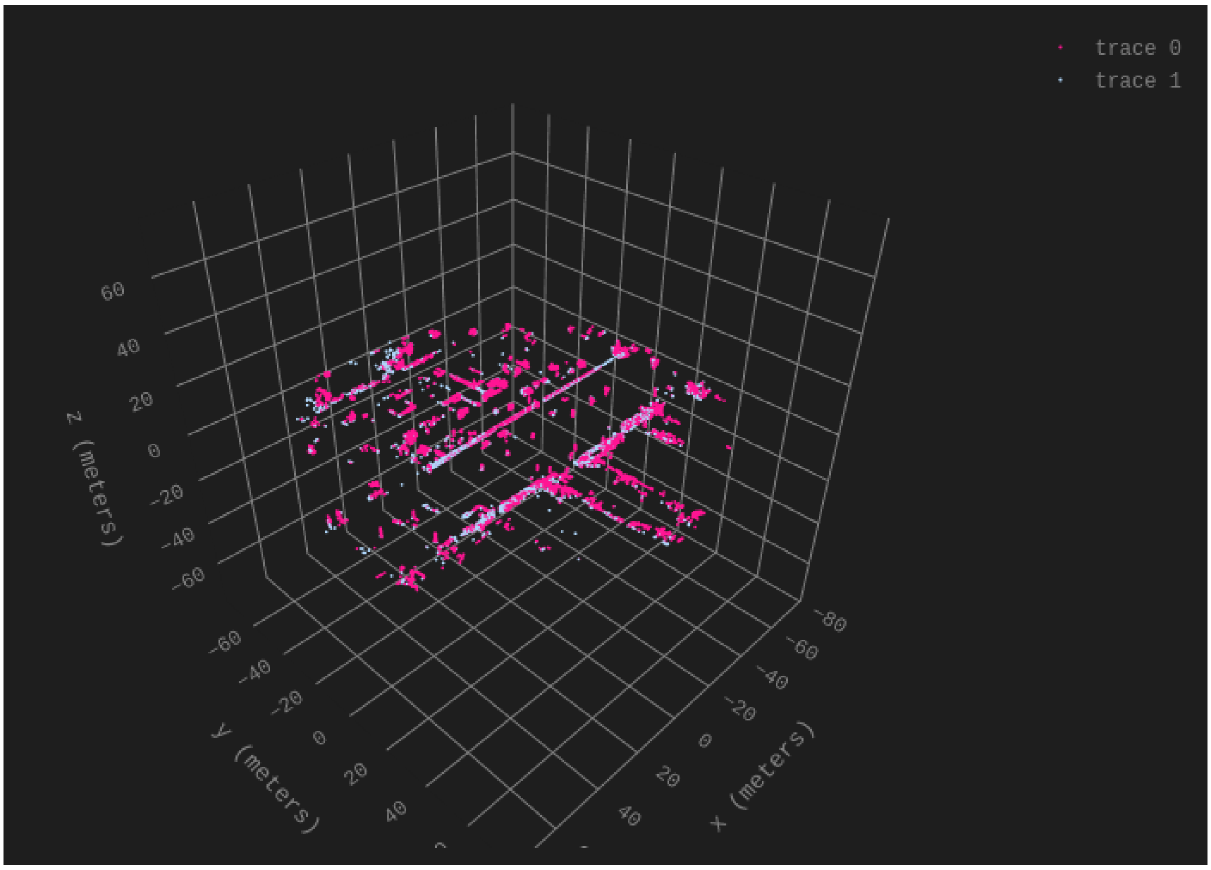


*Figure 25 FilterPC-NearestNeighbors_P2*

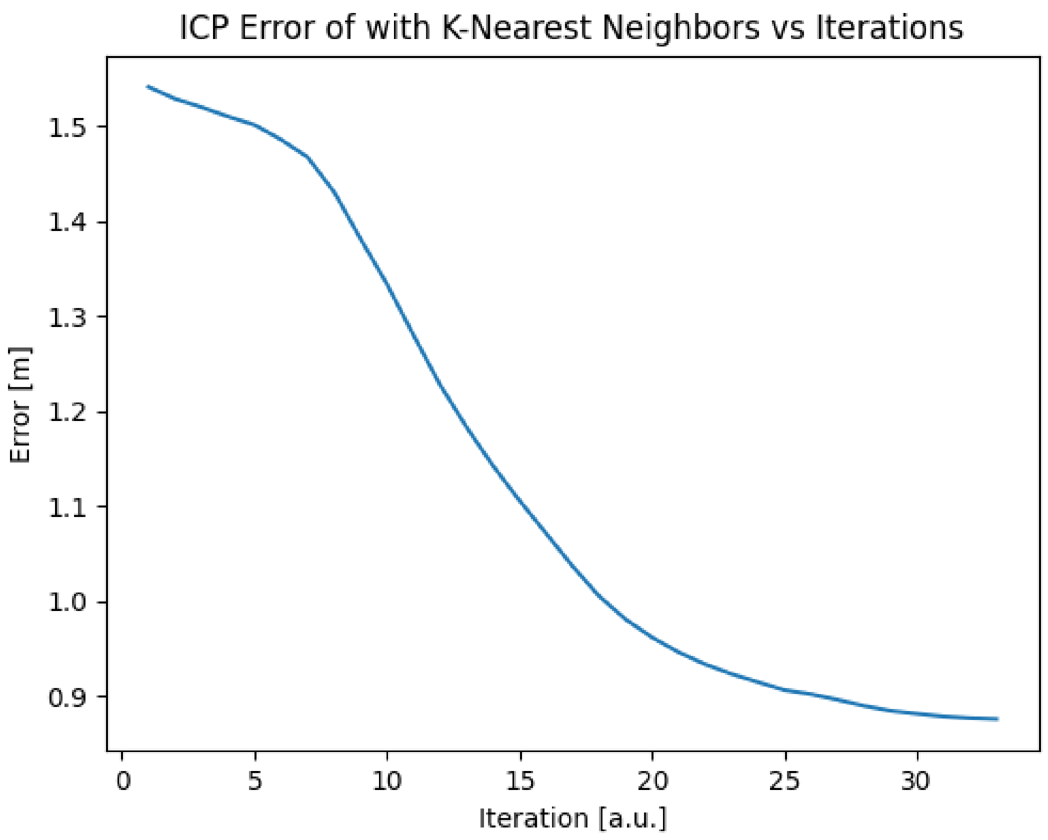


*Figure 26 ICP Error of KNN VS Iterations P2*

We can see that the KNN behaves the same as the K-D-Tree (as can be seen when comparing the 2-animation received), as seen in the previous Point Clouds, we can see in the above image and in the animation that after applying the affine transformation on the 2 PC we receive a good correlation between the two PC with many overlapping point as it should.

We will now compare the results of each of the algorithm on the point clouds:

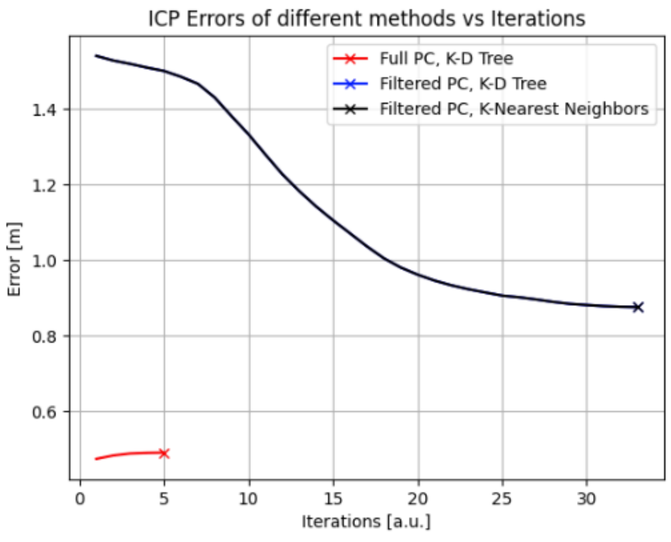


*Figure 27 ICP Errors of different methods VS iterations P2*

```
Final error of K-D Tree is: 0.49061974255956187 [m^2]

Final error of K-D Tree with filtered PC is: 0.8758065150767743 [m^2]

Final error of K-Nearest Neighbors with filtered PC is: 0.8758065150767743 [m^2]

Elapsed time kdree: 3.324seconds

Elapsed time knn: 5.4627 seconds
```

As we can see we got the smallest error for the K-D-Tree on the full PC but as we explained earlier this error is misleading, since its an RMSE most of the points that contribute to this low error belongs to the road and therefore they should not be taken into consideration.

On top of that we can see again that for the filtered PC the K-D-Tree and the KNN overlap meaning we haven't gained anything by applying the KNN but we can see that its runtime is a bit higher than the K-D-Tree runtime as to be expected. We can see that the received final error of this two methods seems higher than the error received from the ICP on the full PC but in practice we can see that the Point Clouds have bigger overlaps than the one seen at the end of ICP on the full Point Clouds, as explained before this small error (seen at the end of IPC on the full point cloud) takes many road points into consideration and therefore the error value itself do not represent the actual difference between the 2 PC in the unique points (i.e points that do not belong to the road).

Overall we can see that for the filtered case we received a good overlap between the 2 PC

## D

The referaed paper presents a novel approach to the Iterative Closest Point (ICP) process that enhances LiDAR-based odometry, providing two significant contributions:

Motion Compensation: The authors have proposed a simple yet effective motion compensation mechanism based on a constant velocity model. This model compensates for the distortions caused by the robot's motion during the scan, making it unnecessary to employ more complex and computationally expensive techniques, such as those used in CT-ICP. The motion compensation approach was evaluated using the KITTI raw dataset, with the results demonstrating that this technique can produce state-of-the-art results.

Adaptive Data-Association Threshold: The paper introduces an adaptive algorithm for setting the data-association threshold (τt), which determines the proximity needed for two points to be considered a match. Instead of using a fixed threshold that needs to be tuned for each different motion profile, this adaptive algorithm adjusts the threshold online based on the motion profile. This results in an equivalent or better performance without the need for manual tuning for each sequence.

The system proposed in this paper makes a few assumptions. The primary assumption is that the point clouds are generated sequentially as the robot moves through the environment. It does not require additional sensor data such as from an Inertial Measurement Unit (IMU), but it does assume that the motion of the robot can be approximated by a constant velocity model, which is used for motion compensation. It also assumes that a robust kernel is used for the ICP calculations. The robustness and effectiveness of these assumptions were tested on different datasets under various conditions, with the system performing well in all instances.

## 3. Visual Odometry

The goal of this section is to implement a simple, monocular, visual odometry (VO) pipeline with

the most essential features:

• Initialization of 2D landmarks,

• Extract key point tracking between two frames,

• pose estimation using established 2D↔ 2D correspondences,

• extract R and T from the essential matrix

• Refine translation value by r ratio

We will be using kitty visual odometry data set, The input (grayscale) Selecting trajectory number 02 from the scenarios. We will then extract the camera calibration intrinsic parameters.In our case we are performing mono VO so we use only camera 0. And will Estimate the motion by only two pairs of images (t, t+1)

We will then calculate the scale factor as the ratio magnitude of GT (L2 norm) and estimated trajectory (magnitude GT is simulated odometer measurements).

### A

Here we will implement monocular visual odometry! (2D→2D). we will calibrate the algorithm for maximum of ~15 meters (Euclidian distance) from the GT (in the first 500 frames). The algorithm pipeline:

- ground truth odometry is extracted and displayed in the following figure:

- for every frame in total number of frames:

• Step 1: extract 2 frames, the current frame and next frame

• Step 2: extract key points and descriptors of both frames. ( performed with sift)

• Step 3: match features between both frames (BF).

• Step 4: get essential matrix using the matched points and remove outliers to the over defined equation using RANSAC

• Step 5: Recover pose by decomposing the essential matrix to rotation matrix and translation vector the pose recovery verifies possible pose hypotheses by doing chirality check. (which means that the triangulated 3D points should have positive depth)

• Step 6: perform scale correction using the ground truth as our data input (in general, it could be imu data or lidar) this is performed because in mono Visual Odomatry the translation is only known up to scale as there is no direct depth estimation.

- We then concatenate the translation vector and rotation matrix from starting point till end to get full trajectory

We will now dive into the detail of each technique (readers that are familiar with these techniques can skip this part):

SIFT ( Ket point finding and descripting)

SIFT (Scale-Invariant Feature Transform), The algorithm is designed to find and describe distinctive, invariant image features that are robust to changes in scale, rotation, and illumination.

Scale-space extrema detection: SIFT starts by constructing a scale-space representation of the input image. This is done by convolving the image with Gaussian filters at different scales to create a series of blurred images called the scale-space pyramid. The scale-space pyramid is built by repeatedly applying a difference-of-Gaussians (DoG) operation, which subtracts adjacent blurred images to enhance local extrema.
Key point localization: At each scale in the scale-space pyramid, potential key points (interest points) are identified by comparing each pixel's intensity value to its surrounding pixels in the scale-space pyramid. Key points are selected based on their local contrast and are also required to be stable across different scales and rotations.

Key point orientation assignment: For each key point, an orientation or gradient direction is computed. A local image gradient histogram is created around the key point, considering the gradients of pixels in its neighbourhood. The dominant gradient orientation is determined, and the key point is assigned that orientation. This step makes the algorithm rotation invariant.
Key point descriptor calculation: In this step, a descriptor is computed for each key point to represent its local neighbourhood. The descriptor captures the appearance and gradient information around the key point. A square region around the key point is divided into smaller subregions, and for each subregion, histograms of gradient orientations are computed. These histograms are combined to form the final descriptor, which is a high-dimensional vector.

BF (Matching):

brute force matcher takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned.

Essential Matrix:

The essential matrix captures the geometric relationship between two camera views of a scene, allowing for the reconstruction of the scene's structure. It encodes information about the relative translation and rotation between the cameras, as well as the internal parameters (intrinsic matrix) of the cameras.

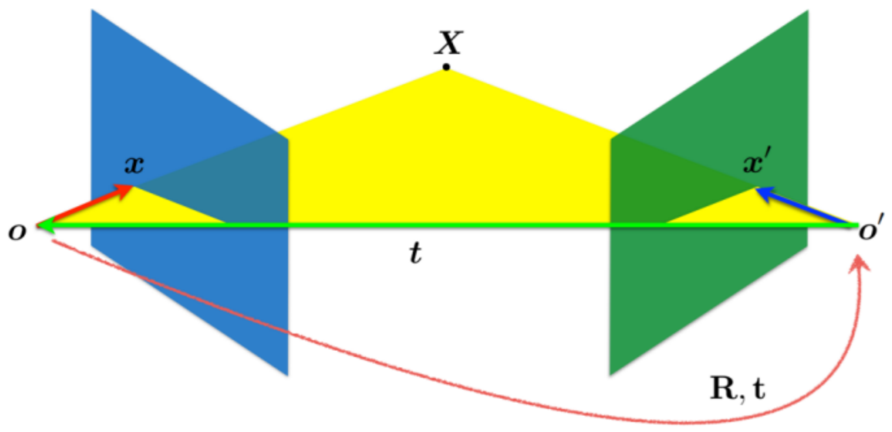


*Figure 28 epipolar geometry*

From this we can derive the epipolar constraint where x, x'(corresponding points), t are coplanar:

rigid motion $\qquad\qquad$ coplanarity

$$\boldsymbol{x}' = \mathbf{R}(\boldsymbol{x} - \boldsymbol{t}) \qquad (\boldsymbol{x} - \boldsymbol{t})^\top(\boldsymbol{t} \times \boldsymbol{x}) = 0$$

$$(\boldsymbol{x}'^\top \mathbf{R})(\boldsymbol{t} \times \boldsymbol{x}) = 0$$
$$(\boldsymbol{x}'^\top \mathbf{R})([\mathbf{t}_\times]\boldsymbol{x}) = 0$$
$$\boldsymbol{x}'^\top(\mathbf{R}[\mathbf{t}_\times])\boldsymbol{x} = 0$$

$$\boldsymbol{x}'^\top \mathbf{E}\boldsymbol{x} = 0$$

The essential matrix is then defined as: $E = R[T]x$ To compute the essential matrix we need at least 5 corresponding points and the more we have the better as we will refine the calculation using ransac algorithm Solving:

$$[P_2; 1]^T K^{-T} E K^{-1} [P_1' 1] = 0$$

where E is an essential matrix, p1 and p2 are corresponding points in the first and the second images, respectively. We will then pass the essential matrix to recover the pose. to recover the relative pose between cameras.

Scale correction:

In order to recover the unkown scale factor we use the true translation distance as our info measure it could have been made better with lidar info and gps info of true distance between two points as well we used the following scale factor

$$scale_i = \frac{\sqrt{((curr\_GT_x - prev\_GT_x)^2 + (curr\_GT_y - prev\_GT_y)^2 + (curr\_GT_z - prev\_GT_z)^2)}}{\sqrt{t_x^2 + t_y^2 + t_z^2}}$$

We will now add the figure showing the distance from the GT at frame 500:

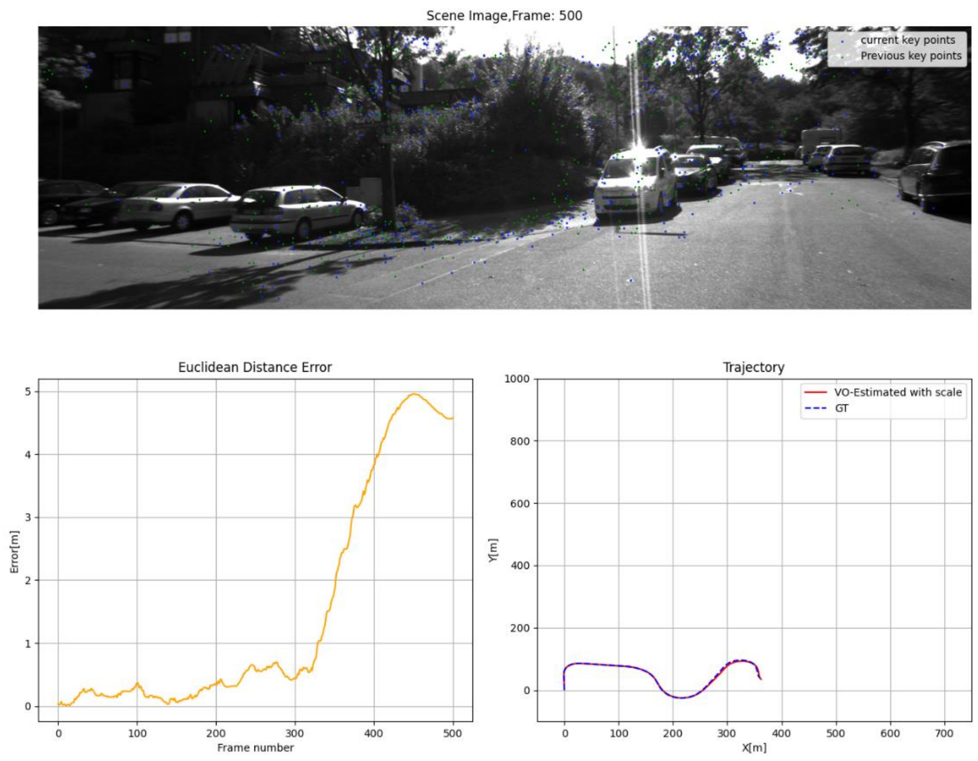


*Figure 29 distance from GT after 500 frames*

As we can see in the above image by implying the algorithm specified above we indeed managed to reach the desired goal of error of maximum Euclidian distance of 15 meters in the first 500 frames.

## B

A. The receives animation is attached to the submission files, for ease of the computer computation the first 500 framed were saved as is and for all the rest of the frames we saved every 4th frame and included the last frame (as Roy instructed in the course forum) . In the animation we can see the original grey scale images, with the each frame containing the features (i.e. the current key points and the previous key points), the ground truth and the visual odometry trajectory and the Euclidean distance error.

B. We will now analyze the results comparing the estimated trajectory to the GT and Identify possible reasons for the drift observed in the results:
first we will add the final image from the video corresponding to the finale frame

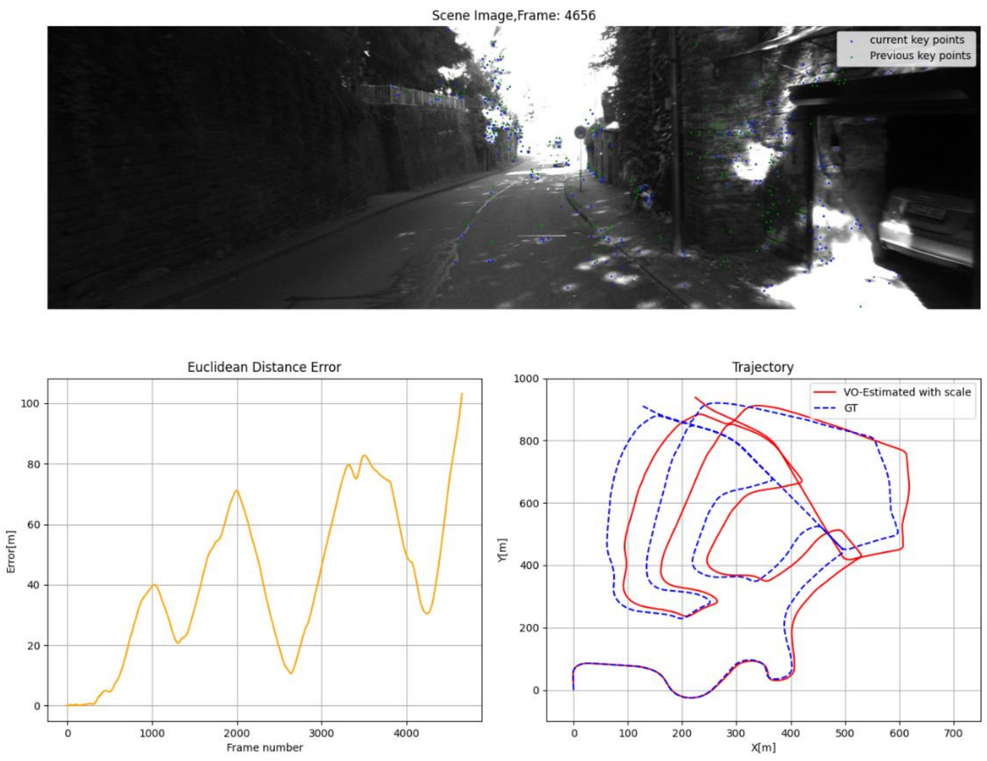


*Figure 30 GT an VO comparison*

As we can see in the video and in the above image the trajectory that the we took is a winding track which makes it harder to track, we can see the indeed starting roughly from the 400th frame the VO estimated trajectory and GT trajectory starts to differ from one another we believe that the reason for that is Accumulated errors: Monocular visual odometry relies on incremental estimation, where each frame's motion is estimated based on the previous frame's estimation. Over time, errors starts accumulate, leading to a drift in the estimated trajectory. Even small errors in feature tracking or depth estimation can have a cumulative effect and cause deviations from the ground truth trajectory. And since that in no point during the algorithm's run we close the loop, meaning we compare the GT value to VO value we have no way of mitigating this drift. Even though we can see areas in the graph were the errors do decrease we refer this decrease to the turning points in the trajectory, meaning that this decrease is more of a fluke than an actual error decrease since that as we can see in the video the errors return after we exit each curve.
Other than that we can see that for each frame the algorithm managed to find good feature points and track them between frames (if it was not the case we would have seen a bigger divergence between the 2 trajectories). In conclusion we can say that the algorithm managed to produce a good trajectory estimation in the first 500 or so frames but later on the algorithm started to diverge and accumulate errors in its trajectory.

## C

We will now suggest 3 mechanisms to improve the results theoretically:

1. Sensor Fusion with External Sensors: One effective approach is to incorporate data from external sensors, such as a GPS, to enhance the accuracy of visual odometry. Sensor fusion

techniques combine the measurements from multiple sensors to obtain a more reliable estimation of camera motion. By fusing visual odometry data with the data recived from the GPS sensor, we can correct for drift and improve the accuracy of the trajectory estimation. Sensor fusion algorithms like Extended Kalman Filters (EKF) can give us improved results.

2. Loop Closure and Global Optimization: Another approach to refine the visual odometry trajectory is to introduce loop closure detection and perform global optimization. Loop closure detection aims to identify previously visited locations in the scene, allowing the system to recognize and correct drift. Once loop closures are detected, a global optimization algorithm, such as Graph-SLAM (Simultaneous Localization and Mapping), can be employed to optimize the entire trajectory by minimizing the accumulated errors. This helps to align the estimated trajectory with the ground truth trajectory.
3. Visual-Inertial Odometry (VIO): Visual-inertial odometry combines the advantages of both visual odometry and inertial measurement units (IMUs) to improve trajectory estimation. IMUs provide measurements of acceleration and angular velocity, which are used to track the camera's motion independently of visual cues. By fusing the visual information from the camera with the inertial measurements from the IMU, VIO algorithms can mitigate the limitations of visual odometry and achieve more accurate trajectory estimation.

## Summary

We will write a short summary for each part of the 3 parts in the project

1. Particle Filter:
   In this part of the project we focused on implementing and analyzing a particle filter for trajectory estimation in robotics. The particle filter demonstrated its ability to handle non-linear and non-Gaussian distributions, effectively incorporating uncertainties in motion and sensor measurements. The analysis showed that the particle filter achieved accurate trajectory estimates, even in the presence of noisy data. However, challenges such as computational complexity and particle degeneracy were identified. To address these challenges, suggestions were made to utilize advanced sampling and resampling techniques, along with adaptive mechanisms for adjusting the particle distribution. By implementing these enhancements, the project aimed to develop more efficient, accurate, and robust particle filters for diverse applications in the field of robotics.

2. Iterative Closest Points:
   In this part of the project we implemented the Vanilla Iterative Closest Points (ICP) algorithm and analyzed its performance on point cloud data. The process involved loading two point clouds and corresponding input images, implementing the K-D algorithm for efficient multi-dimensional search, estimating the transformation via Singular Value Decomposition (SVD), and applying the ICP routine iteratively until convergence. The algorithm found corresponding point pairs, computed the affine transform, updated the total transformation,

and checked for convergence based on the mean square error (MSE). We evaluated  the algorithm's performance on different point cloud datasets, both with and without filtering out road points. The results showed that the algorithm achieved convergence with reduced error when applied to filtered point clouds, indicating improved correlation between the point clouds. Additionally, we compared the performance of K-D Tree and K-Nearest Neighbors algorithms on the filtered point clouds, observing similar results but with varying runtimes. Finally, we discussed a referenced paper that presented motion compensation and adaptive data-association threshold techniques to enhance LiDAR-based odometry using ICP. These techniques demonstrated improved performance and robustness, as evaluated on the KITTI raw dataset.

3. Visual Odometry
   In this part of the project, we implements a monocular visual odometry (VO) pipeline with essential features. We utilized the Kitty visual odometry dataset and focuses on initializing landmarks, tracking key points, estimating pose, and refining translation. The algorithm employs SIFT for key point extraction, BF for feature matching, and the essential matrix for capturing camera relationships. Results show that the estimated trajectory deviates due to accumulated errors and the lack of loop closure. Three theoretical improvements are suggested: sensor fusion with external sensors, loop closure and global optimization, and visual-inertial odometry (VIO). These approaches enhance accuracy and mitigate drift for more reliable trajectory estimation.