VO-project

Han Wang, 21-947-882

Haoran Chen, 21-952-270

The main pipeline follows the instruction provided. In addition, we implemented three methods in the feature-extracting and feature-matching step. The first method is self-written function of harris feature detection and patch matching. The second is self-written function of KLT tracking. The third is built-in matlab computer vision tools. Generally, the third method shows the better result.

1. Pipeline

In the initialization step, we manually select two bootstrap frames, extract harris corners from both frames and try to match those corners. From those matched points, we implement the estimate essential matrix function to set up the initial localization of the camera pose. This function includes RANSAC to filter out outliers. The initial camera pose of the second frame is stored in the state variable “state”. The matched points in the second frame are selected as keypoints “P” and also stored in “state”. The harris corners that are not matched but with high harris response are selected as candidate points “C”, and we compute the corresponding “F” and “T”. “C”, “F”, and “T” are stored in “state”.

In the process frame step, we compute the current state based on the information of previous state. First, we try to match the keypoints “P” in the previous state and compute the current camera pose by RANSAC localization while removing outliers in the matched keypoints. Then, we also try to track the candidate points “C” in the current frame and update the candidate points. This update procedure includes using reprojected error to filter out false matched points and using the angle checking to select candidate points “C” as keypoints “P” in the current frame. After that, we select new candidate points “C” with high harris response and these points are selected from regions that are not too near the keypoints “P” or the matched candidate points. Finally, we construct the current state variable “curr_state” for the next process frame step.

2. Additional feature

1. Refine the camera pose.

In the process frame step, after adding new keypoints “P” from the candidate points “C”, we reimplement RANSAC localization to get a refined camera pose of the current frame and filter out outliers to get keypoints “P” with better quality.

2. Pyramid KLT tracking.

In the KLT tracking method we implement, we add a pyramid KLT tracking function. When the KLT tracking fails to track enough number of keypoints, the pyramid KLT tracking enhances the tracking ability by tracking at different scales.

3. Implementation

3.1 Harris-based Pipeline

Taking the ‘Parking’ datasets as an example, Figure 1 demonstrates part of the result when adopting Harris detector and matching methods. The pipeline works well when processing the first few frames, but it fails to go on due to the number of landmarks triangulated is getting less and less.

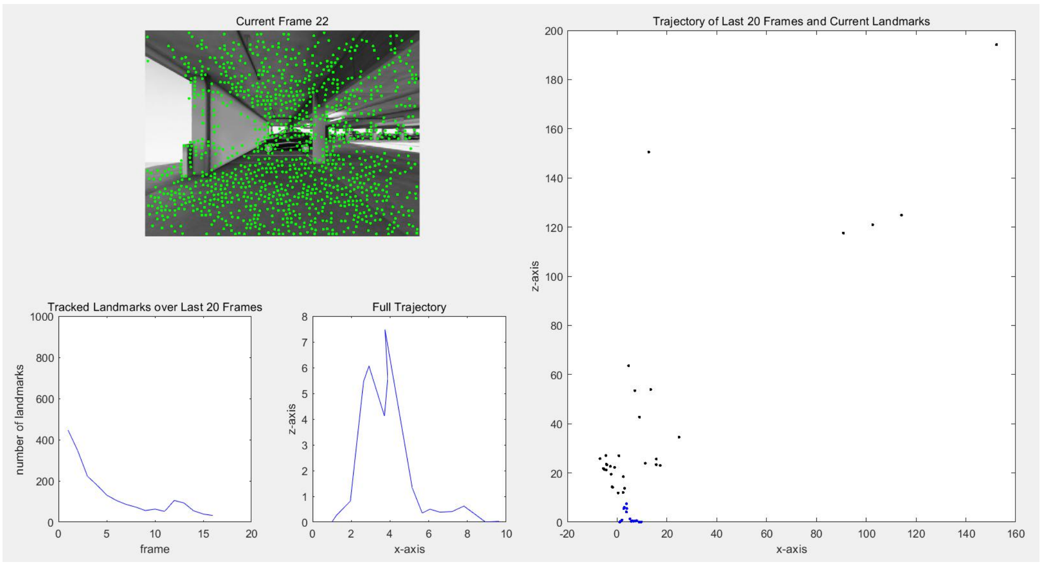

Figure 1 Harris-based pipeline implementation

In Figure 2, there exists so many false matching, and therefore so many outliers will negatively affects the subsequent steps like RANSAC, Triangulation.

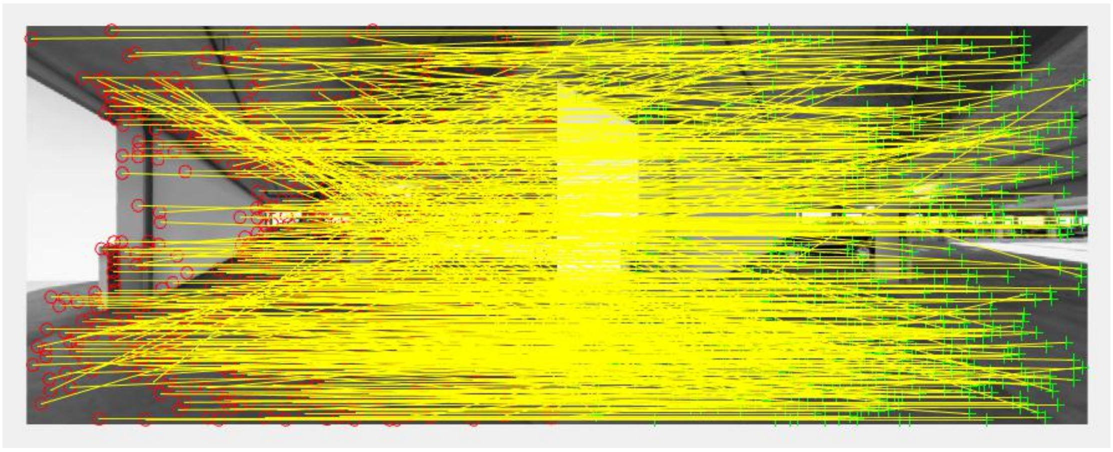
Figure 2 Harris matching result

3.2 KLT-based Pipeline

Based on Harris-based pipeline, a KLT-based pipeline is further implemented to achieve more accurate matching. In Figure 3, it is not difficult to see that the KLT typically gets a more satisfying matching, so the pipeline implement will be better than Harris-based pipeline (Figure 4).

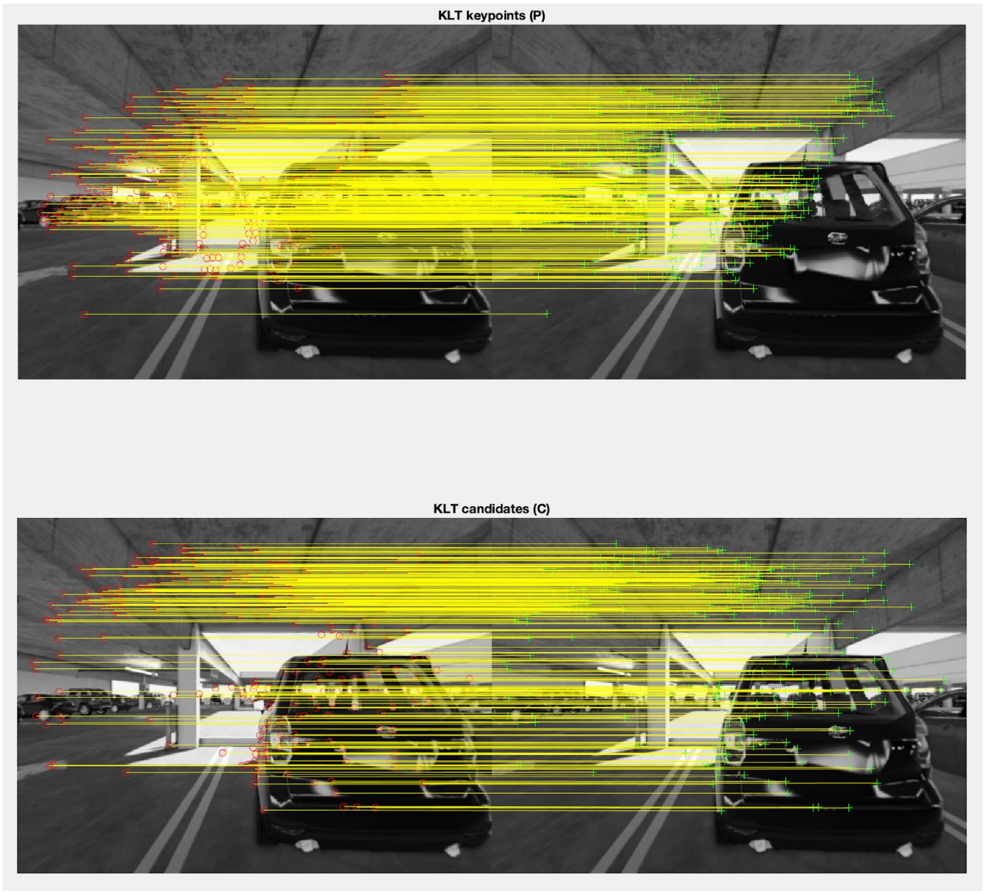

Figure 3 KLT tracking result

This method is able to process more frames and demonstrates less error. But what the problem we encountered is that the pipeline will suddenly exhibit irreversible errors in frame 29 and fail quickly in the next few frames. To eliminate the cause of failure, we further replace the regular KLT with the **Pyramid KLT**. The result (Figure 5 & 6) shows that Pyramid KLT is more computation costly and fail to work better although it can successfully tracks more features.

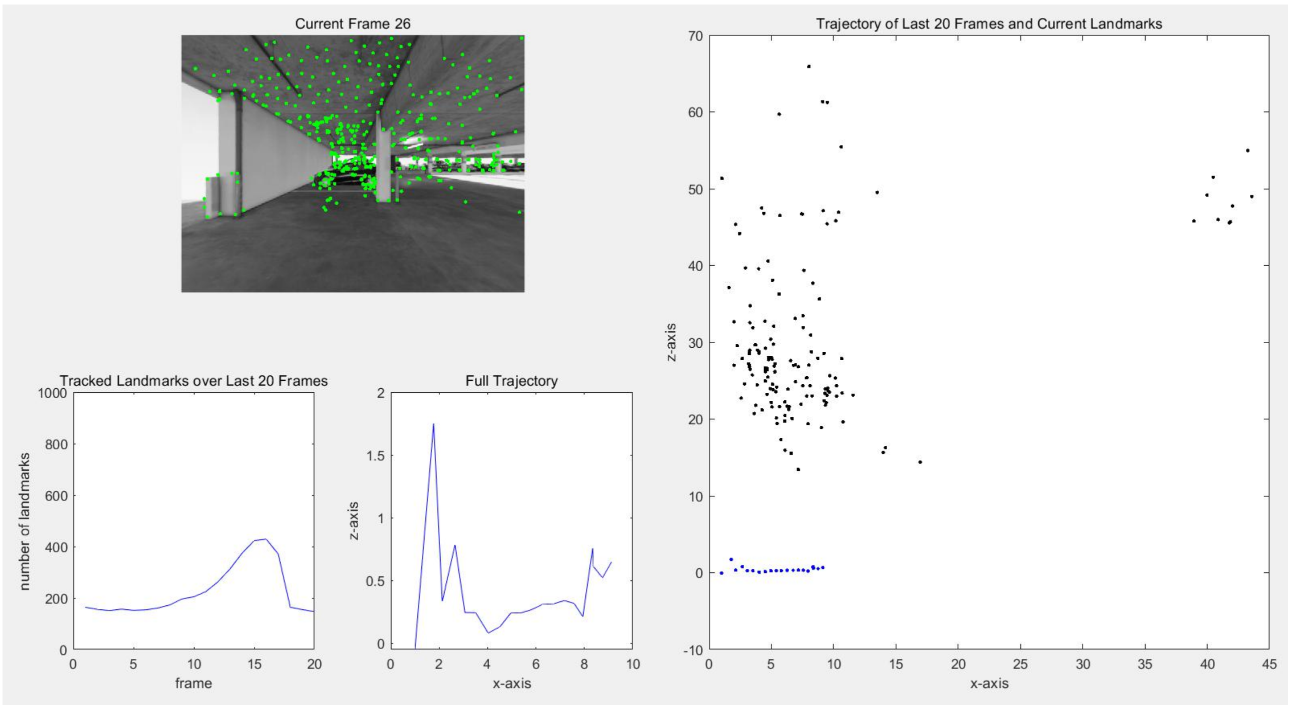

Figure 4 KLT-based pipeline implementation

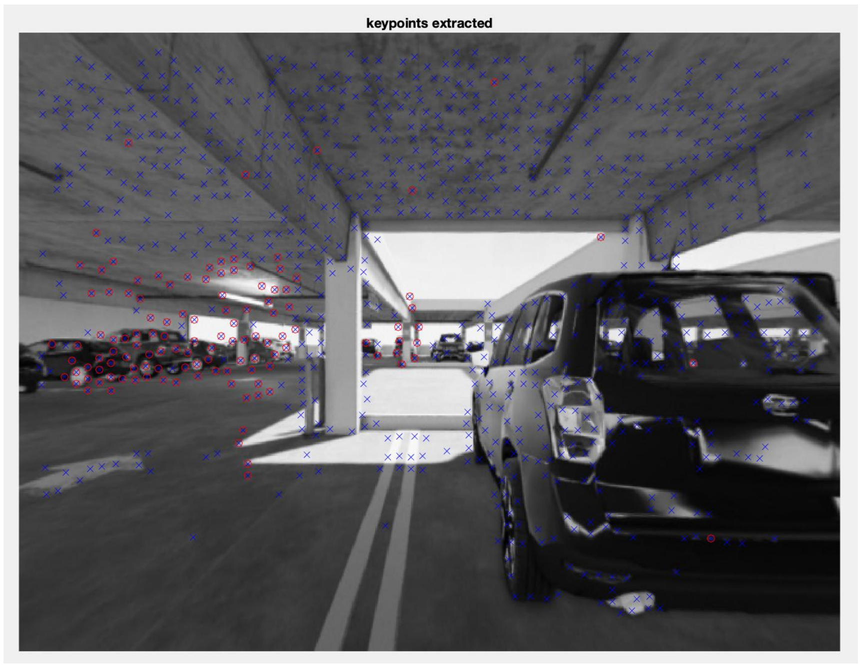

Figure 5 Tracked points(red) with regular KLT

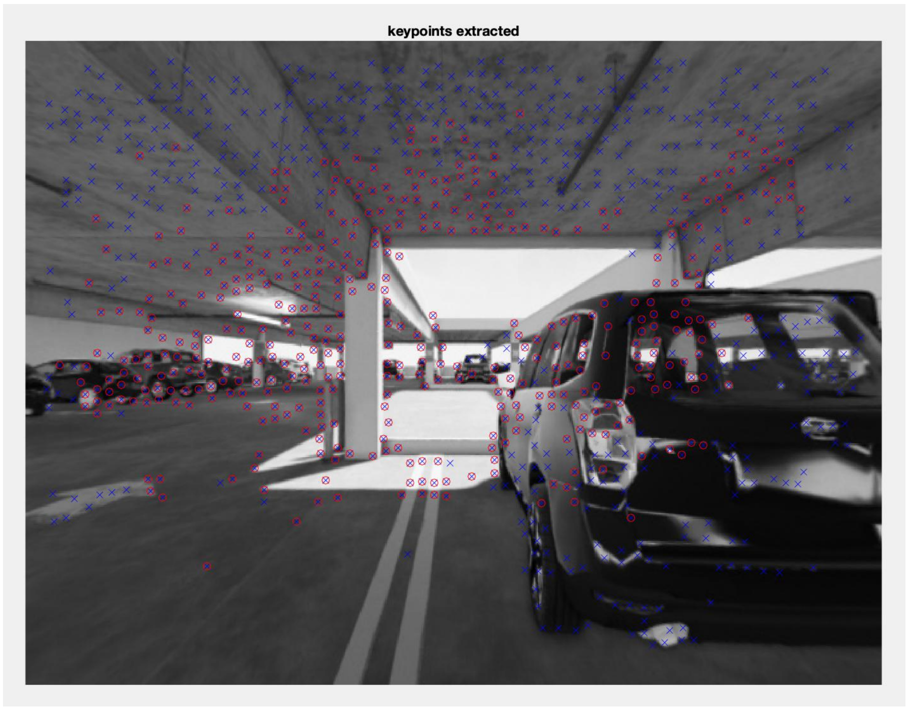

Figure 6 Tracked points(red) with Pyramid KLT

To make the pipeline more precise, **bundle adjustment** is performed with matlab function bundleAdjustment(). Compared to regular-KLT pipeline, the pipeline works better and can process up to frame 59 without large error after performing bundle adjustment (Figure 7).

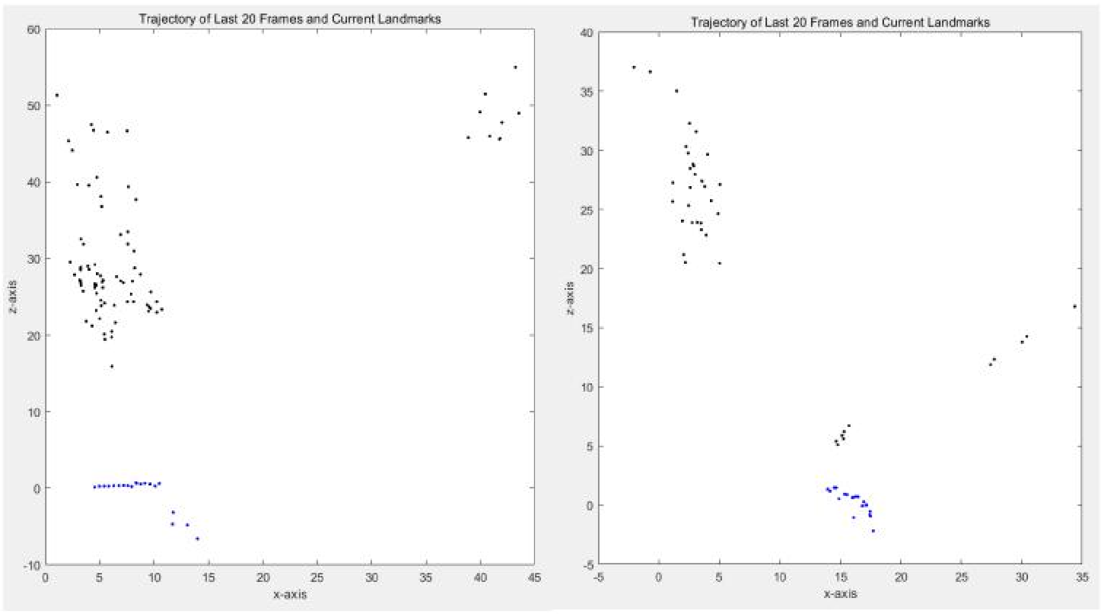

Figure 7 KLT-based pipeline(left, frame 30), KLT-based pipeline+BA(right, frame 60)

3.3 Matlab-func-based Pipeline

Both of the Harris-based pipeline and KLT-based pipeline is self-written function and is based on the solution codes provided in the class. And after applying bundle adjustment, the pipeline works a litter better, we consider the possible reason for failure is that the algorithms is relative rough and not accurate Thus, we replace some self-written function with MATLAB inner functions including Harris detector, KLT tracking, PnP and Triangulation, but remain the general pipeline unchanged. Finally, the third pipeline works evidently better as expected and is able to process the whole datasets when replacing the previous ‘Triangulation’ function.

Figure 8 shows the final implementation. When processing about the first 230 frames, the error in z-position is less than 5 units. Also, Figure 9 & 10 show two intermediate results, showing the pipeline still works fine even if processing so many frames.

However, the pipeline does not perform such well when working on the Malaga and KITTI datasets. A possible approach to solving this, also to better processing Parking datasets, might be replace all the self-written functions with MATLAB inner function and perform local optimization like Bundle Adjustment and Pose Graph Optimization..

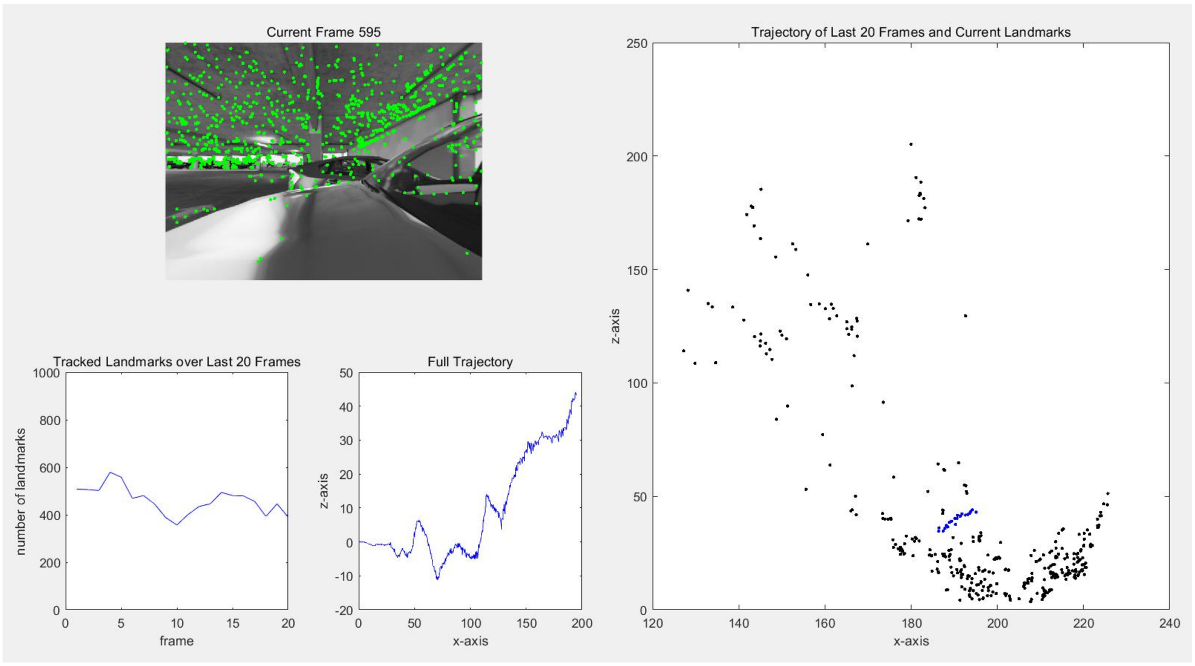

Figure 8 matlab-func-based pipeline implementation

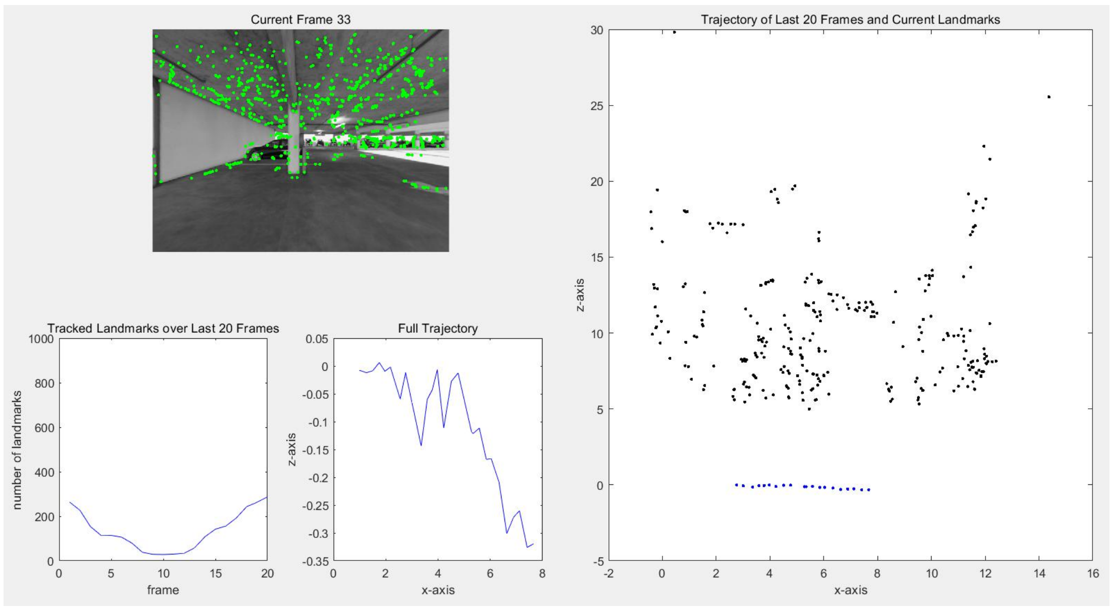

Figure 9 matlab-func-based pipeline intermediate result-1

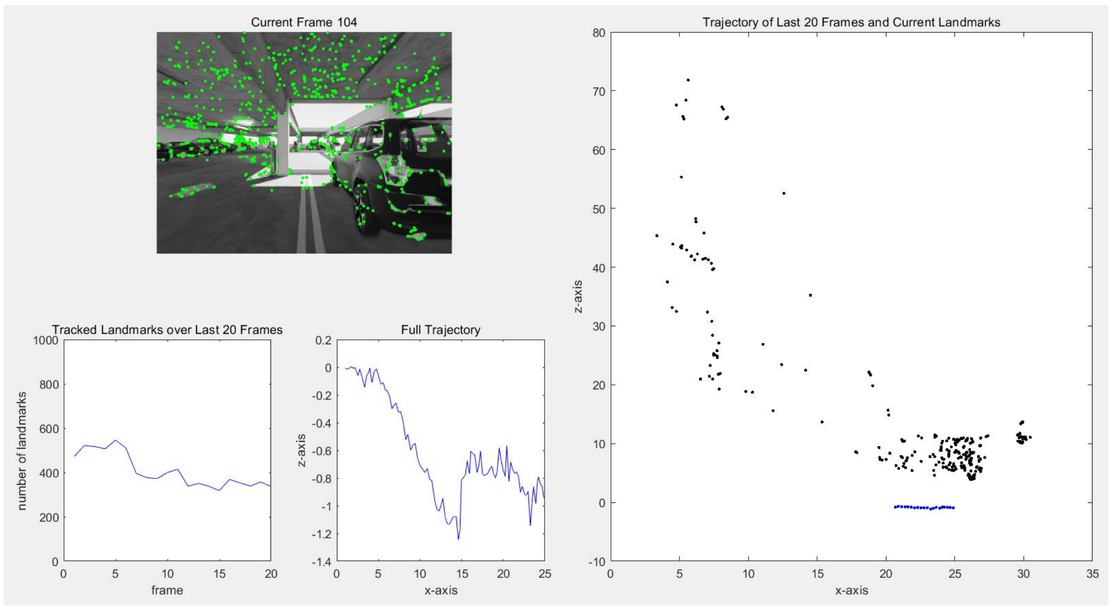

Figure 10 matlab-func-based pipeline intermediate result-2