3DCV 2022 Final Project - Visual-Based Localization

R09922030 沙佳哲　R10922051 梁瑜軒　B07902060 趙雋同

June 2, 2022

Abstract

The Global Positioning System (GPS) is one of the most popular localization method. However, it is not accurate enough for applications such as self-driving cars. Several methods were developed to increase precision of localization, but with the cost of extra base stations or external expensive sensors. Vision-based localization utilizes monocular camera, which are easily available in many cases such as car dash cameras. Previous works of vision-based localization have shown to be able to provides high precision localization at low-costs. Nevertheless, they are generally too slow for real-time applications. In this final project, we aim to reproduce results of previous works. Furthermore, we analyse the bottleneck of inference speed and attempt to speedup with recent methods. We conducted experiments on self-collected data and provide detailed discussions.

1 Introduction

The Global Positioning System (GPS) is the most widely used technology for localization. However, the precision of GPS is approximately 3-20 meters [Dri07, MKtH06]. Furthermore, the GPS is not capable of distinguishing between traffic lane and highways. In applications such as AR natural guidance, or self-driving cars, high-precision localization is required, and GPS cannot fulfil this need.

In order to increase the precision of localization based on GPS, many technology have been developed. [RDL06] uses external base stations to increase the precision of GPS. [YCWX21] utilizes RADAR on LiDAR maps to achieve end-to-end localization. Although these methods can achieve high precision localization, they are not widely deployed due to the high cost issue. Hence, it is important to develop an accurate but inexpensive solution.

Vision information is commonly seen in modern day vehicles and mobile devices, such as dash cameras on vehicles and cameras on smart phones. These information could be utilized for localization, which are otherwise wasted in most cases, and comes with low-cost. One of the pioneers of vision-based localization is [CWW+16]. Their method can be divided into training phase and inference phase. In the training phase, they construct a 3D point cloud model from videos by using Structure from Motion (SfM) technology with the SIFT [Low04] feature descriptor. They then compress the model with structure preserving compression to reduce the number of points in the model. During inference phase, whenever an inference image arrives, SIFT will be detected and computed from the image. 2D-to-3D matching will be followed to match the image features to the 3D model. Finally, pose estimation methods can be applied to get the global localization result.

Despite high precision localization is achieved by [CWW+16], inference speed is still unsatisfying. Their method is not applicable to applications that requires real-time inference. In our final project, we first try to reconstruct their algorithm. Then, we analyse the inference time bottle neck and try to speedup inference by incorporating recent methods. We experimented on a self-collected dataset, and provide detailed observations and discussions.

The remainder of the paper is organized as follows. We review related studies in Section 2. In Section 3, we describe the methodology of our approach. Implementation details and experimental results are presented in Sections 4. Finally, we draw the conclusions in Section 5.

2 Related Work

We review several related works in this section. Our main difference between [CWW+16] is by using different Structure from Motion (SfM) tools, different feature descriptors, and different pose estimation method. We will introduce them briefly in the following sections.

2.1 Structure from Motion (SfM)

Structure from motion is a technology that can construct 3D point clouds out of images. It starts by matching local features between image pairs. Then it estimates the 3D point position and camera pose from the image pair(s) that have matching feature points. Finally bundle adjustment [WACS11] is applied to preserve global consistency.

There are many different branches of SfM, and one of the most popular one is incremental SfM. The main concept of incremental SfM is to first construct the 3D point cloud model with a couple of images. Then *incrementally* register new images into the model. Two mainstream tools for incremental SfM is VisualSfM [Wu13] and COLMAP [SF16]. VisualSfM is used in [CWW+16]. However, we found out that COLMAP is more robust in model construction compared to VisualSfM empirically. Thus, we decided to use COLMAP in our final project.

2.2 Local Feature Descriptors

Local feature descriptors are widely used to match points between images, and many applications such as SfM and image registration depends on it. One of the most widely used local feature is SIFT [Low04]. SIFT has shown robustness in different image scales and is still competitive to modern methods. However, SIFT detection and computation time of SIFT is a major issue. SIFT cannot achieve real-time inference speed in many applications. Many following works such as SURF [BTG06] or ORB [RRKB11] have been developed to speedup feature extraction time. Nevertheless, they comes with the cost of lower accuracy and results in poor performance in many applications.

With the development of deep learning (DL), many DL-based local features have been developed. SuperPoint [DMR18] is one of them, and has shown superior performance compared to traditional local features. In addition, with the help of hardware such as GPU and the parallel nature of deep neural network (DNN), SuperPoint can achieve real-time feature extraction easily. In our final project, we experimented with both SIFT and SuperPoint, and compare the differences of them in speed and localization performance.

2.3 Pose Estimation

Many algorithms have been developed to solve the pose estimation problem, including DLT [MNT04], EPnP [LMNF09], and many others. DLT simplifies the non-linear PnP problem into a linear problem and solves it directly. It can be easily extended to many points but is less accurate. EPnP transforms the original coordinate system to Barycentric coordinate system. The points of the original coordinate system can be represented by four control points. The PnP problem can then be transformed into a linear problem and be solved efficiently. DLT was deployed in [CWW+16]. EPnP has shown to be faster and also more accurate method to solve the pose estimation problem. Thus, we deployed EPnP in our experiments.

3 Method

Our method is divided into two phases: 1) training phase and 2) inference phase. In the offline training phase we construct 3D point cloud model from videos/images. Additional compression of the model will also be conducted in the training phase. In the online inference phase, we try to estimate the camera pose of a given inference image.

3.1 Training Phase

The training phase is conducted offline, and is less time-sensitive. It is mainly composed of two parts: 1) Model construction and 2) Model compression. The details are provided in this section.

3.1.1 Model Construction

We use COLMAP for 3D point cloud model construction. Our training image is of 1920x1090 resolution. Moreover, we try to resize training images into 960x540 in order to reduce model size.

In feature extraction phase, we use SIFT and SuperPoint to extract features. SIFT is the default method of COLMAP, which is known as a scale invariant feature extraction. SuperPoint, a deep-learning method, was proposed in recent years as feature extraction for huge speedup improvement from SIFT. In COLMAP construction, ImageReader.single_camera flag is set to 1, since our dataset is captured from a single camera. And we use OpenCV camera model for COLMAP construction.

After COLMAP construction, since a single point in point cloud may be reference as keypoint of multiple images, we use mean of those descriptors to compress point cloud size, speeding up the next step of model compression.

3.1.2 Model Compression

Model Compression is essential to achieve real-time inference. Without compression, the bottleneck of inference speed would be at matching points between the inference image features and the model features. We inherit the compression method from [CWW+16]. The compression algorithm is composed of two parts: 1) Structure detection 2) K-cover point selection. We will describe the algorithm in details here.

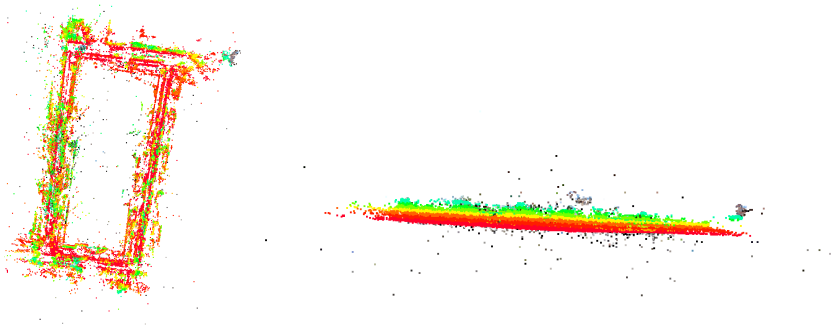
Structure detection: Structure detection is first performed before we select points. The reason we are performing structure detection is that we would like to preserve the main structures of the model after compression, and select points that distributes uniformly across all structures. The structures we are aiming for are *planes* and *lines*. We deploy a simple RANSAC algorithm [YF10] with a *one-at-a-time* strategy (detecting one plane or line at a time). The RANSAC algorithm for plane detection is described in Algorithm 1. Detection for line is similar to plane, we modify the number of points from 3 to 2, and calculate line accordingly.

By using the RANSAC algorithm, we are able to detect planes and lines. However, we discovered that the planes and lines do not behave as we expected. Our test data covers a large area and contains extremely many points. As a result, the detected planes are almost all parallel to the ground (Fig. 1). To face this issue, we proposed two solutions: 1) Local voxel detection and 2) Birdview detection.

1) Local voxel detection: In this approach, we simply divide the model in to small voxels, and detect structure with the RANSAC algorithm locally in each voxel. The results (Fig. 2) show that planes parallel to the ground is still likely to be detected. Moreover, the size of the voxel is hard to determine.

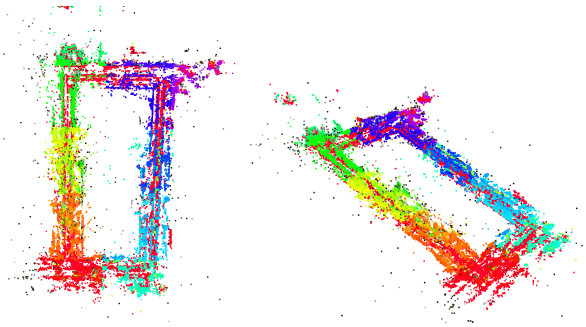
2) Birdview detection: We found out that most structure we would like to detect are walls of the buildings, which are all perpendicular to the ground. We also discovered that the first plane is always the ground, as it covers the most points. This brings us to the birdview detection approach, where we detect the first plane with the RANSAC algorithm, project the remaining points to it, and finally perform line detection on the projected points. This approach results in structure like walls in the model (Fig. 3), and is closer to our expectation. AS a result, this approach is selected for our experiments.

The planes and lines detected are denoted as r_l ($l = 1 \cdots L$), where L is the total number of planes and lines. The remaining points that do not belong to any planes or lines are grouped in to a single category $L + 1$.



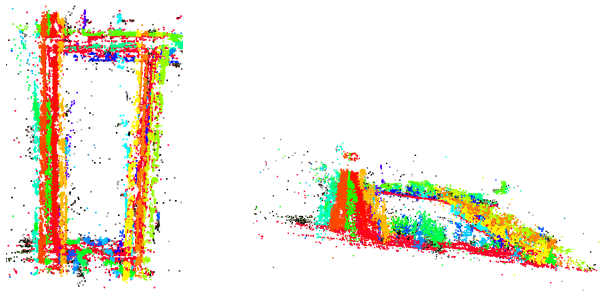
(a) Birdview (b) Sideview

Figure 1: Detected structure with the RANSAC algorithm.



(a) Birdview (b) Sideview

Figure 2: Detected structure with the local voxel detection.



(a) Birdview (b) Sideview

Figure 3: Detected structure with the birdview detection.

K-cover point selection: After structure detection, we have to solve the K-cover problem. That is, to ensure that each training image would preserve at least K feature points after compression. We first assign weights w_i for each structure r_l as follows:

$$w_i = \begin{cases} \sigma_l, & if P_i \in r_l \\ \sigma_L + 1, & otherwise, \end{cases} \tag{1}$$

with

$$\sigma_l = |P_i | P_i \in r_l \forall i| / N, \tag{2}$$

$$\sigma_{L+1} = |P_i | P_i \notin r_l \forall i \forall l| / N. \tag{3}$$

Based on the weights computed, a weighted K-cover problem is formulated. The minimum set K-cover problem is NP-hard. Therefore, we use a greedy algorithm to solve it efficiently. We calculate a score for each point and select the maximum score in each round. The score is computed from two terms, the weight w_i we calculated above, and visibility $v(P, c_j)$, which represents if a image contains the point or not.

$$P_s = \arg \max_{P_i \in \mathbf{P}} \sum_{j=1}^{m} w_i v(P_i, c_j) \tag{4}$$

with

$$v(P, c_j) = \begin{cases} 1, & if P \in c_j, \\ 0, & otherwise, \end{cases} \tag{5}$$

After each round, the weight will be updated to ensure points will be selected over all structures. If a point in r_l is selected in this round, the weight will be divided by 2 for the next round. By greedily selecting points with this score, the compressed model would preserve points that are more likely to be seen and points that are in large plane or lines.

Algorithm 1 RANSAC for plane detection

$bestSupport \leftarrow 0; bestPlane \leftarrow NULL$
$bestStd \leftarrow \infty; i \leftarrow 0$
$\epsilon \leftarrow 1 - inlier\_prob$
$N \leftarrow round(\log(1-\alpha)/\log(1-(1-\epsilon)^3))$
while $i \leq N$ **do**
  $j \leftarrow$ pick 3 random points
  $pl \leftarrow$ plane from j
  $dis \leftarrow$ distance of all points to pl
  $s \leftarrow$ find $abs(dis) \leq t$
  $st \leftarrow$ Standard-deviation of s
  if (length(s) > $bestSupport$) or (length(s)=$bestSupport$ and $st < bestStd$) **then**
    $bestSupport \leftarrow$length(s)
    $bestPlane \leftarrow pl; bestStd \leftarrow st$
  end if
  $i \leftarrow i + 1$
end while

3.2 Inference Phase: Camera Relocalization

In the online inference phase, a sequence of query images is given. The goal is to estimate the camera pose of the query image with respect to our constructed 3D point cloud model. We split the inference procedure into three sub-tasks: Feature Extraction, Point Matching and Pose Solving.

1. Feature Extraction: We extract keypoints and descriptors from the query image. Extraction method has to be consistent to those used to construct the model.

2. Point Matching: We tried to match descriptors between model and image. Besides the intuitively brute force method, we also tried kd-tree since descriptors are all high dimensional features.

3. Pose Solving: Multiple camera pose computation method is applied for solving camera position, P3P, PnP, AP3P and EPnP. We can further exclude outliers and unsatisfied reprojection error poses with RANSAC algorithm.

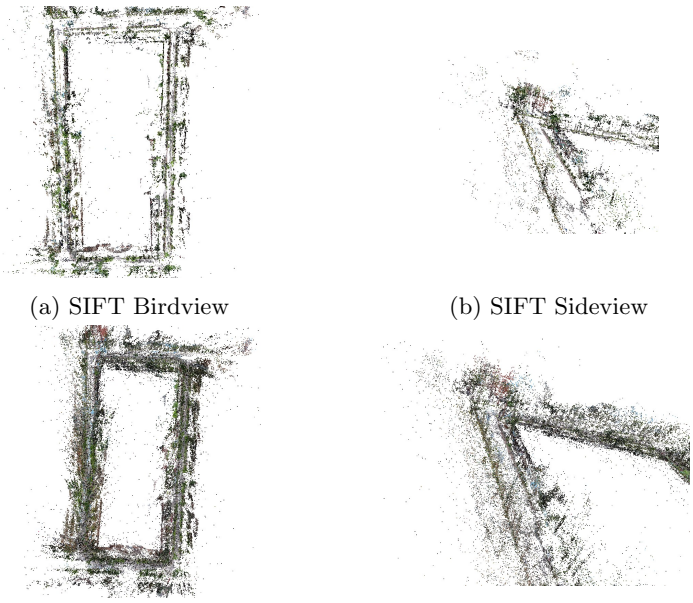
Without any further improvement, our inference performance does not exceed 5 FPS. Moreover, we analyze the computation time of each inference procedure and found that point matching was mainly the bottleneck since there are huge amount of descriptors with the uncompressed model. For the compressed model, feature extraction is the bottleneck. Hence we applied two methods to tackle on the problems above.

1. Image Resizing: Intuitively, resizing images helps improve speed of feature extracting. While accuracy of the pose would be reduced owing to lack of 3d-to-2d correspondences.

2. Local Matching: To reduce numbers of 3d point candidates from model, an naive method is to restrict distances between previous pose and points occurs since poses between two continuous frame wouldn't be far from each other. Nevertheless, this method does not work since most inliers at a distance will be excluded and outliers behind camera perspective remained. Finally we excluded absolute outliers by simulated a pyramid according to rotation and translation information from previous frame and reduce the computation for matching significantly.

4 Experiment

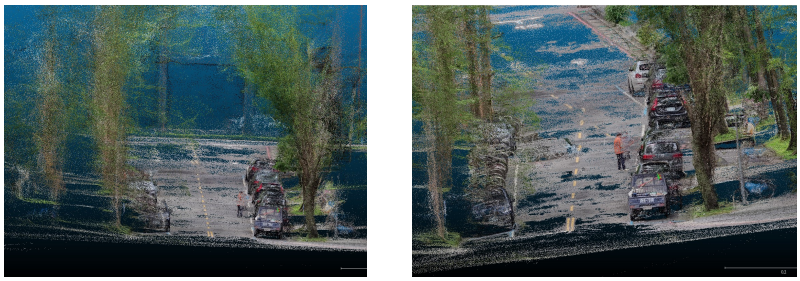
We present the results of our experiments in this section. Observations and detailed discussion are provided in each section.



(a) SIFT Birdview (b) SIFT Sideview

(c) SuperPoint Birdview (d) SuperPointSideview

Figure 4: Model construction using SIFT and SuperPoint.



(a) SIFT (b) SuperPoint

Figure 5: Loop part of constructions.

4.1 Experimental Settings

Video data is recorded using Samsung Galaxy S20FE with resolution 1920x1080 using 30 FPS. Training video is converted into images every 10 frames, and contains 641 images. Inference video is converted into images every 7 frames, resulting in 568 query images. Our project runs on Windows 10 OS with Intel i5-11400 CPU, 32G RAM and Nvidia RTX 2060.

4.2 Model Construction

We use all 641 training images with size 1920x1080 as our model construction data, the runtime of COLMAP procedure is about 5 hours. We also attemped to downsample image data to 960x540 pixels, which spends about 2 hours for COLMAP procedure. There is no obvious difference of runtime using different feature extraction methods.

Fig. 4 shows the result of COLMAP construction. Construction using SIFT obtains a smooth and loop shape in geometric range. However, construction with SuperPoint feature extraction performs worst while loop occurs (Fig. 5).

Since the amount of matched feature points using SuperPoint is much more than using SIFT, the point cloud in SuperPoint method is denser than SIFT method. Result in the difference of models' size. And downsampling data images' size has little reduction on model size.

4.3 Model Compression

We used the birdview detection for structure detection as mentioned in Sec. 3.1.2 The model size of the original model and after compression is presented in Table 1. The size of the model is reduced significantly after compression (0.5% for SuperPoint at K=20!). The inference results are still reasonable after compression, and compression provides huge inference speedup. One thing to notice is that the model size after compression is independent to the original model size. This is due to the compression algorithm design. The compression algorithm aims to preserve K points of each training image. Thus, the preserved number of points is only related to the training images and will be roughly the same regardless of the original model size.

	Original	K=200	K=100	K=50	K=20
SIFT	120 MB	13.3 MB	6.3 MB	3.1 MB	1.3 MB
SuperPoint	265 MB	13.4 MB	6.1 MB	3.0 MB	1.3 MB

Table 1: Model size under different values of K

4.4 Inference: Camera Relocalization

Inference speed and relocalization success rate under different settings are presented in Table 2 and Table 3, respectively. The first column indicates which feature is used, and the relative size of image used in inference. We compare the performance of inference with uncompressed model and compressed model with different K. Note that the success rate reported is merely the rate of camera pose being properly estimated without high relocalization error. We have observed some cases where the reprojection error is low, but the localization is completely incorrect. The success rate is also not capable of evaluating the localization precision. As we have no ground truth trajectory of our inference data, it is impossible to calculate error. We inspected the results with our bare eyes and most cases seems to be accurate. To sum up, the relocalization success rate gives us a rough idea of how our implementation performs, but is likely to be over-optimistic.

An example of our implementation result can be found in Fig. 6. Information of inference time can be found on the top left corner, where "Detection" corresponds to feature extraction time, "Filtering" corresponds to point of local matching filtering time, "Matching" corresponds to 2D-to-2D matching time, and "Solve Pos" corresponds to pose estimation time. With these information, we are able to identify the bottleneck of inference time in different settings. We could find the localization results in the bottom left corner. The red pyramids represents the camera pose. The lines shooting out from the camera represents the matches found by 2D-to-3D matching. The green lines are the ones identified as inliers after pose estimation, while the blue lines are the outliers. As we could see from the results, the localization are highly accurate. More inference videos could be found in our online GoogleDrive[1].

From the results, we could first notice that the inference speed has increased significantly after compression, and as K decreases, the inference speed is further increased. Reducing image resolution also benefits inference speed by a lot. However, we have to be careful of how much to reduce the image resolution, as the performance (both success rate and precision) drops significantly if we reduce the image resolution too much.

In feature extraction, SuperPoint has better speed compared to SIFT. The reason is that SuperPoint can be speedup with parallel computing and the help of GPUs. In comparison, SIFT is computed on CPU in our experiments, and might still be able to catch up in speed with techniques like SiftGPU [Wu07]. Another thing worth mentioning is that under different compression rate, SuperPoint is still accurate when K is extremely small (K=20), while SIFT requires at least K=100 to perform normally. This shows that the robustness of SuperPoint keypoint detection. Under different image resolution, we noticed that the success rate does not degrade much for SIFT, and degrades significantly for SuperPoint, especially with 1/3 image resolution. This shows that SIFT is indeed scale invariant, and SuperPoint is more sensitive to image size.

Because SuperPoint and SIFT are all high dimension features, kd-tree searching performs a bit better than brute force and the value for ratio test is set to 0.75 empirically. Also, due to fact that

[1]Inference videos: https://drive.google.com/drive/folders/1QuVm8hQINQjTym3AJXzGTbwrSLWh7nco

Figure 6: Inference result examples

global matching has poorer speed but higher accuracy than local matching, we found out that switching to global matching is necessary when local matching not working. Besides, angle of view for simulating perspective pyramid is required to bigger because there might be rapid turn in intersections. To cover all potential inliers the angle is choose to be set 90°

As for pose solving, we found out PnP, P3P, AP3P and EPNP have nearly same performance. Since the major bottleneck occurs at feature detection and point matching, we did not conduct much experiments on pose solving.

The inference performance is highly related to environment change. For example, we first use data from homework 3 as our inference data but turns out accuracy is too poor. By visualization, we figured out that it is caused by un-matching of 2d-to-3d descriptors since we still got lots of keypoints and descriptors from feature extraction stage. But keypoints and descripors of rainy day (homework 3 video) and shinny day (video record by ourselves) seemly vary a lot. The potential solution to this issue might be storing various models constructed under different weather conditions and select the best fit model according to the weather conditions in inference time.

	Original	K=200	K=100	K=50	K=20
SIFT (1/1)	0.52 FPS	1.64 FPS	2.00 FPS	2.29 FPS	2.46 FPS
SIFT (1/2)	3.20 FPS	8.00 FPS	9.15 FPS	10.67 FPS	10.67 FPS
SIFT (1/3)	7.11 FPS	16.00 FPS	21.33 FPS	21.34 FPS	21.34 FPS
SuperPoint (1/1)	0.94 FPS	4.52 FPS	5.01 FPS	5.27 FPS	5.33 FPS
SuperPoint (1/2)	2.21 FPS	16.00 FPS	18.01 FPS	21.05 FPS	21.34 FPS
SuperPoint (1/3)	0.77 FPS	32.01 FPS	32.01 FPS	32.01 FPS	32.01 FPS

Table 2: Inference Speed under different settings

	Original	K=200	K=100	K=50	K=20
SIFT (1/1)	100%	100%	97.71%	91.90%	62.32%
SIFT (1/2)	100%	99.65%	99.12%	97.54%	88.20%
SIFT (1/3)	98.94%	98.77%	97.01%	89.97%	59.16%
SuperPoint (1/1)	100%	100%	100%	99.82%	99.12%
SuperPoint (1/2)	94.54%	99.30%	96.83%	91.37%	77.47%
SuperPoint (1/3)	80.81%	61.09%	51.76%	38.38%	20.42%

Table 3: Relocalization success rate under different settings

5 Conclusion

In this final project, we successfully reproduced previous work of vision-based localization. Our main focus is to attempt to improve the inference speed without sacrificing too much localization precision. Many details of the algorithm has to be adjusted properly to achieve reasonable results. Our implementation reached around 16 FPS with the best settings, and the localization precision are still accurate. We also observed some key differences between traditional local feature SIFT and modern deep learning method SuperPoint. We've put a lot of effort into this project and experimented with various methods, many of which are not presented in this report. Below are some potential extension directions, which we might try in the future.

To scaling up out model, we must face performance problem in inference phase. Especially on point matching stage. To address this problem, we should probably split the model into several parts. Ideally one should not able to observe any point from other models except the model he/she located. In this way we could switch model when one cross to the overlapping area between two models. Furthermore, some points that are being matched are actually points behind wall or trees, this might result in poor localization precision and also slow down infernece speed. These points might be able to be removed if visibility information is provided. One potential solution is to use the normal vector of points while model construction. By using the normal vector, we are able to filter out some points that are not visible to the current camera.

To address the issues of missing trajectories, which is caused by incorrect pose calculated by camera relocalization, temporal smoothing, such as using Kalman filter, might help us on predicting next frame's pose. Furthermore, the predicted pose might help us improve both speed and accuracy of local matching since local matching applied previous frame's pose to simulate field of view. Using predicted pose would fix the deviation between current pose and previous pose, especially under the circumstances where drivers turns sharply or speed up.

6 Devision of Work

The model construction part is mainly done by Mike (梁瑜軒). The model compression part is mainly done by Chia-Che (沙佳哲). The inference part is mainly done by Anderson (趙雋同). The presentations and report are done collaborately by all members.

References

[BTG06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[CWW+16] Kuan-Wen Chen, Chun-Hsin Wang, Xiao Wei, Qiao Liang, Chu-Song Chen, Ming-Hsuan Yang, and Yi-Ping Hung. Vision-based positioning for internet-of-vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 18(2):364–376, 2016.

[DMR18] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 224–236, 2018.

[Dri07] Ted Driver. Long-term prediction of gps accuracy: Understanding the fundamentals. In *Proceedings of the 20th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2007)*, pages 152–163, 2007.

[LMNF09] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155–166, 2009.

[Low04] G Lowe. Sift-the scale invariant feature transform. *Int. J*, 2(91-110):2, 2004.

[MKtH06] Marko Modsching, Ronny Kramer, and Klaus ten Hagen. Field trial on gps accuracy in a medium size city: The influence of built-up. In *3rd workshop on positioning, navigation and communication*, volume 2006, pages 209–218, 2006.

[MNT04] Kaj Madsen, Hans Bruun Nielsen, and Ole Tingleff. Methods for non-linear least squares problems. 2004.

[RDL06] Anette RietDorf, Christopher Daub, and Peter Loef. Precise positioning in real-time using navigation satellites and telecommunication. In *Proceedings of The 3rd Workshop on Positioning and Communication (WPNC'06)*. Citeseer, 2006.

[RRKB11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[SF16] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4104–4113, 2016.

[WACS11] Changchang Wu, Sameer Agarwal, Brian Curless, and Steven M Seitz. Multicore bundle adjustment. In *CVPR 2011*, pages 3057–3064. IEEE, 2011.

[Wu07] Changchang Wu. A gpu implementation of scale invariant feature transform (sift). *http://www. cs. unc. edu/˜ ccwu/siftgpu/*, 2007.

[Wu13] Changchang Wu. Towards linear-time incremental structure from motion. In *2013 International Conference on 3D Vision-3DV 2013*, pages 127–134. IEEE, 2013.

[YCWX21] Huan Yin, Runjian Chen, Yue Wang, and Rong Xiong. Rall: end-to-end radar localization on lidar map using differentiable measurement model. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[YF10] Michael Ying Yang and Wolfgang Förstner. Plane detection in point cloud data. In *Proceedings of the 2nd int conf on machine control guidance, Bonn*, volume 1, pages 95–104, 2010.