On-Device Visual Odometry: SuperPoint

Authors: Yuqing Qin, Yukun Xia

yuqingq, yukunx

December 14, 2021

1 Motivation

Visual odometry and mapping are very important to mobile robot navigation, autonomous driving, and virtual reality. As the fundamental steps, feature detection, description, and matching have been well developed during past decades. Among those approaches, the classical computer vision methods, such as ORB and SIFT, are still popular and commonly used in recent applications.

In recent years, learning-based methods for feature detectors and descriptors have had the potential to have better performance than the classical methods. On certain tasks, eg. homography estimation, a deep learning model, SuperPoint, outperforms SIFT, ORB, and LIFT. However, learning-based methods are typically more computationally expensive. On high-end GPUs (i.e. Titan X), the real-time operation is possible (i.e. SuperPoint: ∼70 FPS), but these GPUs are hardly accessible to mobile applications, eg. AR glasses. To the best of our knowledge, it has not been carefully explored whether lower computing edge devices, such as Jetson Nano, are capable of efficiently running these deep learning detectors and descriptors. Therefore, the main goal for our project is to experiment with the learning-based method (SuperPoint) on the edge device (Jetson Nano).

2 Task Definition

The project goal is to optimize the SuperPoint model (i.e. light backbone, quantization), deploy the optimized models on edge device (Jetson Nano 4GB), and compare it with the classical methods (i.e. SIFT, ORB) on different metrics. In detail, our project mainly focuses on the below three tasks:

- Retraining the SuperPoint model with different backbones to shrink the model size
- Apply quantization after training and deploy the model with different configurations on Jetson
- Develop auxiliary modules to benchmark the performance for each engine and compare with the classical approaches

3 Related Work

3.1 Keypoint Detection and Description

Our project is mainly based on the SuperPoint paper [1], and our baselines are the two pre-trained SuperPoint models [2, 3] and OpenCV's implementation [4] of ORB [5] and SIFT [6]. The SuperPoint paper novelly designed two convolutional decoders for keypoint detection and description, and the first step of the SuperPoint is to train the neural network on a small-scale synthetic dataset, where the ground truth of the interest points is easy to be calculated. To improve its performance on real images, the authors put forward a self-supervision training method, *Homographic Adaption*. For images from the dataset, e.g., MS-COCO [7], homography transformations can be randomly sampled, and the heat maps of the detection layer across different transformations will be aggregated as the learning target of the current epoch. As for the classical baseline methods, ORB comprises the Oriented FAST detector and Rotated BRIEF descriptor, achieving two orders of speed increasing relative to SIFT, while SIFT has been the state-of-the-art among all the classical methods. Both methods are much more mature than the learning-based alternatives, and in industry, ORB and SIFT are respectively widely used for online and offline visual SLAM applications.

3.2 Neural Network Backbone Choices and Optimization

To optimize the model size, and deploy the deep learning model on mobile devices, people have proposed different variations on the model structure. The most well-known one for shrinking the model size is the MobileNet[8]. The main idea is to use the depth-wise separable convolution to replace the plain convolution. It could effectively decrease the model's number of parameters by dividing the standard convolution into two separate convolutions: depth-wise convolution, and point-wise convolution. SqueezeNet[9] is another popular model variations to shrink the model size. The main idea for SqueezeNet is to use a 1x1 convolution layer to squeeze the feature map first, and use a mix of 1x1, and 3x3 convolutions to expand to the original output size. By applying 1x1 convolutions, it could decrease the input channels to the 3x3 convolutions. However, the decreased model size usually leads to a lower performance score. ResNet18[10] is another light model structure that only contains 18 layers including the last fully-connected layer. At the same time, it could also achieve high performance due to its residual block. In our project, we would mainly focus on applying these three model structure strategies to the SuperPoint backbone and compare the model size with the original one.

4 Main Results and Discussion

4.1 Experiment Settings

4.1.1 Baselines

In our project, we considered two kinds of baselines: SuperPoint, and classical approach as shown below. There are two SuperPoint pre-trained models with slightly different model structures. Their model size is similar but performance is slightly different. Another main category for our baseline is the classical approach. Here, we used ORB and SIFT as our baselines.

- Pretrained SuperPoint model (by Magic Leap)
- Pretrained SuperPoint model (by third party) [1]
- Classical Approach: ORB, SIFT

4.1.2 Model Configurations

As we mentioned in the task definition, our project mainly focuses on four different kind of configurations for the SuperPoint model:

- Model Backbones[2] (6): Pretrained SuperPoint (by Magic Leap), Pretrained SuperPoint (by third party), MobileNet v1, MobileNet v2, SqueezeNet, ResNet18
- Input Resolution (3): 120x392, 240x784, 360x1176

[1]This model was transformed from the weight file in this link. The model was trained on MS-COCO, while it's unknown whether the Magic Leap one has ever leveraged KITTI.

[2]Due to the limitation of the intermediate data resolution, not every layer of the backbones were used.

- Quantization Precisions (2): FP32, FP16 (using TensorRT)
- Input Batch Size (2): 1, 2

Also, we compared the SuperPoint model variations with the classical computer vision approaches as shown below:

- Classical Approaches (2): ORB, SIFT
- Three Input Resolution (3): 120x392, 240x784, 360x1176

Therefore, in total, we have 72 engines for the SuperPoint family, and 6 engines for the classical approach to experiment on.

4.1.3 Hardware

The Hardware we used is Jetson Nano 4GB and our laptops(RTX 2070). Our laptops are mainly used for training and measuring the performance level score (rotation error, translation error). Jetson is used for benchmarking the other evaluation metrics(i.e. latency, power).

4.1.4 Evaluation Metrics

We evaluated our 78 engines on KITTI sequences with the below 6 metrics. Only the Translation and Rotation error is evaluated on our own laptop since the speedrunning on our laptop is faster, and the evaluation result is the same as Jetson. Other metrics are evaluated on Jetson Nano 4GB.

- Translation(%) and Rotation Error(deg/m)
- Number of Parameters
- Power Usage (J)
- Number of Inference Runs with 10Wh Budget (a smartphone battery)
- Carbon Emissions (lbs)
- Latency (ms)

4.2 Results

4.2.1 Number of Params

After retraining the SuperPoint model with four different backbones, we validate their performance on the HPatches dataset and also output their number of parameters as their model size. The below table summarizes the validation performance of the retrained models and their corresponding parameter count.

From the table, we could easily draw the conclusion that MobileNet could effectively decrease the model size by half from the baselines. In the meantime, it also hurts the performance(i.e. detector metric, and descriptor metric). After applying the strategy mentioned in MobileNet v2, which is using the expansion and residual block, the performance increases to a similar performance level as the baselines, but the model size is larger than the MobileNet v1. The model used SqueezeNet strategy performs even better than the one with MobileNet v2 both on the performance score and the model size, but its model size is still larger than the one with MobileNet V1. The model that used ResNet18 as the backbone is larger than anyone else since the original backbone structure only contains 8 convolution layers as its feature extractor, but the ResNet18 contains about 16 convolution layers. This layer difference can also be seen from the model size shown in the table.

Table 1: Validation Performance and Number of Parameters

	Detector metric		Descriptor metric		
Model	Repeatability	Localization error	mAP	Matching score	**No. params**
SP_pretrained1 (Magic leap)	0.606	1.14	0.81	0.55	**1,300,865**
SP_pretrained2 (Open Source Repo)	0.63	1.06	0.77	0.44	**1,304,067**
SP_MobileNetv1	0.61	1.19	0.74	0.42	**754,318**
SP_MobileNetv2	0.625	1.107	0.769	0.443	**949,838**
SP_SqueezeNet	0.627	1.101	0.764	0.442	**847,939**
SP_Resnet18	0.639	1.13	0.76	0.459	**1,942,147**

4.2.2 Performance under Budget

We also evaluated our 78 engines on their power draw as we deployed them on Jetson. We used the command-line tool to directly extract the average energy (J) and convert this number to kWh, then we calculated the number of inference runs under a smartphone battery (10wh) using the equation (# of inference runs = 10wh / energy). We plotted the Performance under Budget with the rotation error and translation error which were evaluated on KITTI. The below plots are color-coded based on different model configurations.

Overall, from Figure 1 and 2, we can see that with smaller resolution (red), we have larger inference runs with larger error, while for the large resolution(blue), the error and inference runs are relatively smaller. Also, with large resolutions, the points are close to each other, but for the small resolution, it has a much larger variance between the points. This observation is reasonable since with small resolution, the information/feature contained in the image is limited, so it is much harder to extract features and performance variants a lot compared to larger resolution inputs. On the other hand, with more features shown in the large resolution, the performance for different model configurations is very stable.

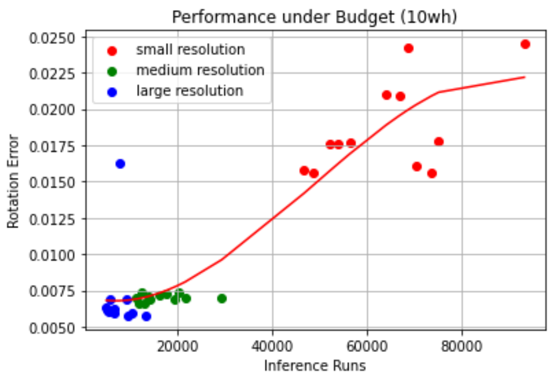


Figure 1: Performance (Rotation Error) under Budget (10wh): Different Resolutions

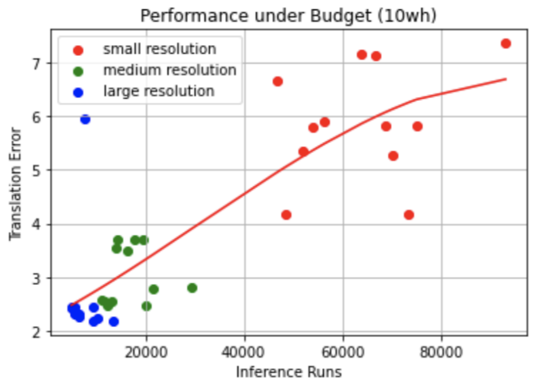


Figure 2: Performance (Translation Error) under Budget (10wh): Different Resolutions

Figure 3 and 4 are the same plot but with different colors. In Figure 3, we compared different SuperPoint models with the classical approaches (i.e. ORB, SIFT). Different from what we see in the Number of Parameter table, this plot suggests that ResNet18 as the backbone is more energy efficient compared to others when the input resolution is larger. Even with the medium resolution, ResNet18 also wins over other backbones as well as the baselines. However, in the small resolution area, the pre-trained model beats others with much larger inference runs but with a smaller error. Another comparison is done between the SuperPoint family and the classical approaches. SIFT (red) performs almost worse than every SuperPoint model, especially worse than ResNet18. With similar error performance, the ResNet18 could run much more inferences within the budget. ORB(purple) is another classical approach we are comparing here. It is the most energy-efficient approach which is proved by the point in the top right corner. However, its error is also the largest among all of the methods shown in the plot. Even in the larger resolution, ORB still cannot beat the ResNet18 one. Overall, SuperPoint with ResNet18 is the best one among both the SuperPoint family and the classical approaches.

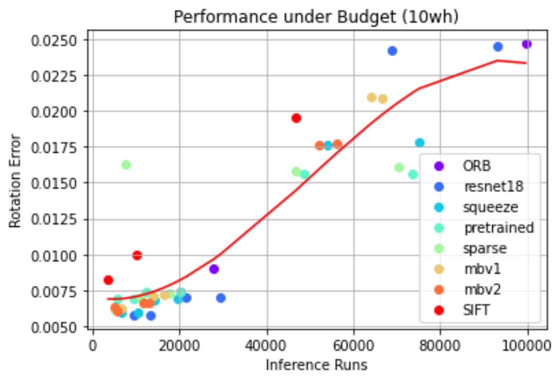


Figure 3: Performance (Rotation Error) under Budget (10wh): Different Model Backbones

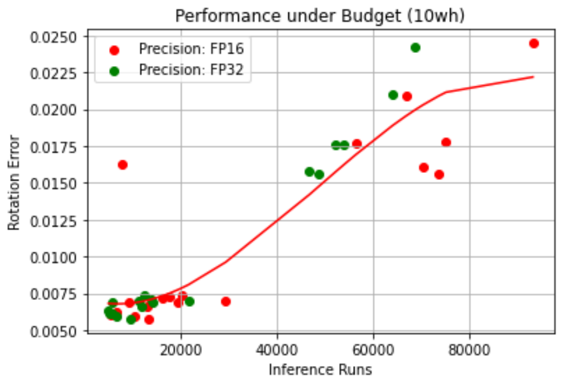


Figure 4: Performance (Rotation Error) under Budget (10wh): Different Precisions

Figure 4 compares the different quantization precisions. We did the post-training quantization through TensorRT. We first converted our trained model to onnx and then converted it to trt engine. From the plot, it shows that for large and medium resolution areas, the precision does not matter that much. FP16 performs slightly better than FP32, but their results are very close to each other. However, in the small resolution, the precision matters a lot. Switching from FP32 to FP16, we see the inference runs increase dramatically, which means that for small resolution, the FP16 is more energy efficient.

Figure 3 and Figure 4 are comparing the inference runs with the rotation error. We also plotted the translation error one, which could derive similar conclusions we have drawn from the rotation error. Those translation error plots are attached in the Graph Appendix.

4.2.3 Carbon Emissions

We further calculated the carbon emissions of both the whole training process and each inference, according to the power or energy data discussed in the last section. The formula is based on our lecture slides, shown below.

$$\mathrm{CO_2\ (lbs)} = 1.0677\ \mathrm{(lbs/kWh)} \cdot \mathrm{PUE} \cdot \frac{\mathrm{t\ (s)} \cdot \mathrm{power\ (W)}}{1000} \tag{1}$$

, where the carbon emission factor 1.0677 (lbs/kWh) is read from this link, and PUE (power usage effectiveness) is 1.58. Our calculation results are documented in Table 2, and ResNet18 wins in both inference and training. Considering comparing all the configurations of the SuperPoint models and the classical methods, we list the top 10 greenest ones during inference in Table 3. All of them are using the small input resolution, and the deep models could be almost as green as ORB. Besides, enabling FP16 and having a larger batch size are the two tricks to reduce the carbon emission from the deep models.

Table 2: Carbon Emissions from Inference and Training

Model (backbone)	Energy per Inference (J)	CO2 per Inference(lbs)	Total Training Time (h)	Average Training Power (W)	Total Training Power (kWh)	Total training CO2 (lbs)
MobileNet v1	0.595	1.7647e-7	8.87	146	2.047	1.950
MobileNet v1	0.798	2.3667e-7	10.72	140	2.370	2.260
SqueezeNet	0.413	1.2248e-7	8.83	145	2.023	1.930
ResNet18	0.319	*0.9461e-7*	6.75	148	1.579	*1.506*

Table 3: The top 10 green model configurations

Model / Approach	Batch Size	Precision	Input Resolution	Carbon Emissions (1e-7 lbs)
ORB	N/A	N/A	120x392	1.071
ResNet18	2	FP16	120x392	1.146
ResNet18	1	FP16	120x392	1.369
SqueezeNet	2	FP16	120x392	1.421
Pretrained (1)	2	FP16	120x392	1.451
Pretrained (1)	1	FP16	120x392	1.466
SqueezeNet	1	FP16	120x392	1.488
Pretrained (2)	2	FP16	120x392	1.518
ResNet18	2	FP32	120x392	1.554
MobileNet v1	2	FP16	120x392	1.598

4.2.4 Performance over Latency

Similar to the energy cost section, the relationships between the performance and the latency[3] are visualized in Figure 5 to 9 [4]. Figure 5 and 6 color-codes the results according to the input resolutions, and clearly, the input resolution is the dominant factor here. There's no significant difference in performance between the large and medium resolution clusters, while the errors of the small resolution cluster are much larger than others. Figure 7 illustrates the impact of the model or method selection. Given medium or large resolution, ResNet18 and SqueezeNet are better than others, i.e. having smaller latency and similar performance. However, in small resolution, although ORB and ResNet18 have the smallest latencies, their performances are much worse than the two pre-trained models. In Figure 8, data points with different batch sizes are shown. Here, the effect of the batch size on latency is not significant, and that's probably because the difference between the two batch size choices is not large enough. Lastly, Figure 9 compares the results of different

[3]The latency considers all steps of the visual odometry, including image pre-processing, neural network inference, data post-processing, and transformation solving.

[4]More graphs related to the translation error can be found in the Graph Appendix, from Figure 12 to 14.

model precision. In most cases, quantization slightly reduced the latency, while the performances stay very close. Still, there's an edge case on the open-sourced pre-trained model with large input resolution, where both rotation and translation error jump dramatically.

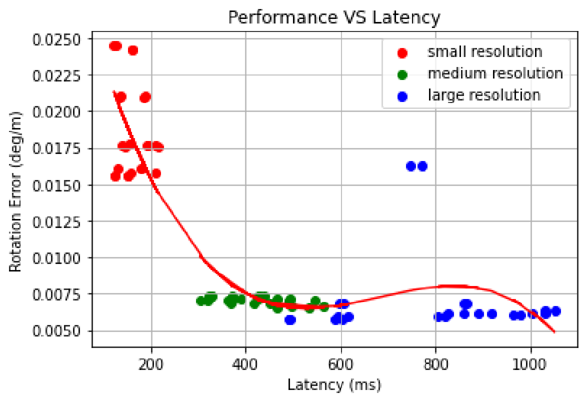


Figure 5: Latency VS Performance(Rotation Error): Different Resolutions

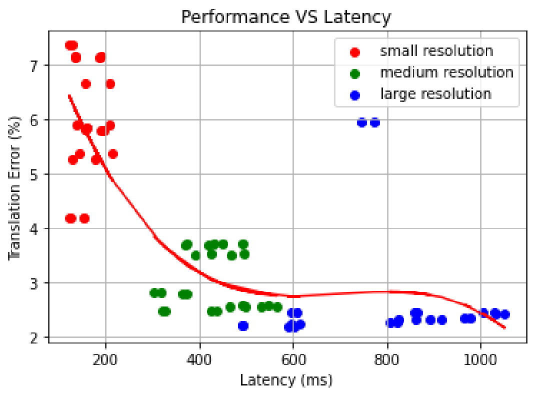


Figure 6: Latency VS Performance(Translation Error): Different Resolutions

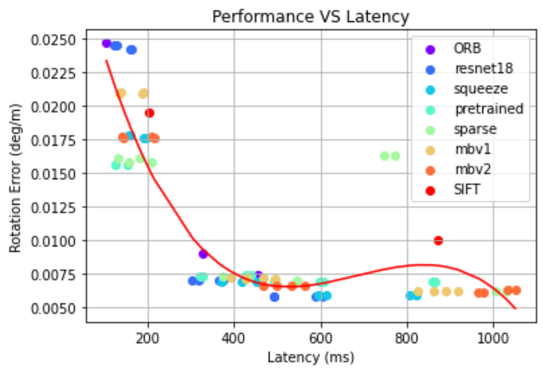


Figure 7: Latency VS Performance(Rotation Error): Different Model Backbones

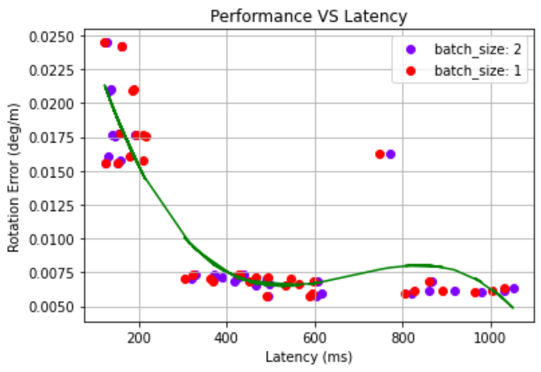


Figure 8: Latency VS Performance(Rotation Error): Different Batch Size

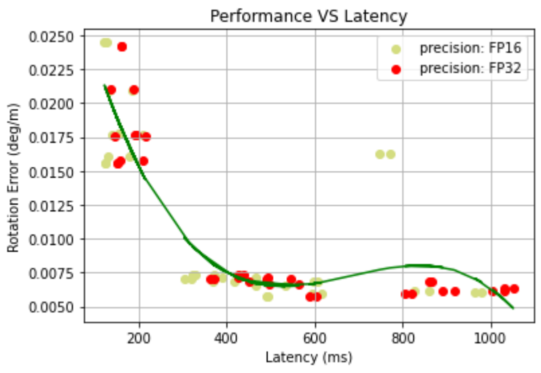


Figure 9: Latency VS Performance(Rotation Error): Different Precisions

5 Challenges

Backbone For the backbones, we considered completely replacing the VGG-style backbone with the MobileNet and SqueezeNet, but it turns out that SuperPoint has to have fixed output resolution for the feature map(H/8, W/8) so that they can be aligned with the pseudo labels generated for COCO. This limits our capabilities to play around with the backbone. To address this problem, we instead only applied the strategies used in MobileNet and SqueezeNet papers to replace the original convolution layers. For MobileNet, we replace the standard convolution layers with the depthwise separable convolution layers to shrink the model size. For SqueezeNet, we instead apply the Fire module to replace the original convolutional layer. However, for ResNet18, we have to truncate the original ResNet18 structure by the point that meets our output resolution requirement.

Latency Test When gathering the latency data for different methods and configurations, we found that the latency results could vary a lot. For example, the latencies of models with 120x392 resolution from long-term testing were about 100∼200ms, while at another time of testing, the results dropped to 50∼100ms. The reason could be that our Jetson Nano had a high chip temperature, eg. 70℃, during the long-term testing, and the chip frequency might be constrained. Besides, even within a short-term continuous test, the latency between inferences also had a small chance to increase a lot. Our model needs to process two images per time step and it was always the first inference to have a larger processing time when we set the batch size to be 1. This means the latency fluctuation is likely to be related to the loss of the cache locality, due to that the visual odometry includes the neural network inference on GPU and post-processing on the CPU. Both of them take roughly half of the running time, and thus the amounts of the computation should be comparable.

Evaluation Our project has a large number of configurations and sequences to be tested and evaluating all of them was quite time-consuming. In total, there're 72 combinations of the neural network models, after different backbones, input resolutions, batch sizes, and quantization precisions are considered. In addition, we have 6 more configurations of ORB and SIFT to be tested. For each configuration, running the whole KITTI dataset takes more than 1 hour. It'd be almost impossible to manually change the configuration parameters and rerun all tests. Therefore we had to spend extra time writing code for long-term evaluations, and it also triggered new issues which were never seen in any short-term test. For example, the program might shut down in the middle of the test for no specific reason, and the problem was not repeatable.

6 Insights

Classical Methods vs SuperPoint Within the available KITTI sequences, we can conclude that in general, SuperPoint is competitive against ORB and SIFT on Jetson Nano, in terms of energy efficiency, latency and performance. The energy efficiency and latency comparison are definitely dependent on the hardware. For example, the SuperPoint inference speed could be significantly slower on a CPU, and the energy consumption would also increase accordingly. But still, it's quite obvious that SuperPoint provides much more uniformly distributed key points than ORB, and the inference speed could be close to ORB when choosing ResNet18 or SqueezeNet as the backbone. Compared with SIFT, SuperPoint performance is slightly better in high and medium resolutions, while the inference speed is much faster. However, this comparison may not be fair, as SIFT was run on CPU, but SuperPoint was run on GPU. In order to draw a more rigorous conclusion, it's necessary to also accelerate SIFT with GPU, but that's not available in the latest OpenCV library (version 4.5.4).

Backbones Surprisingly, ResNet18 is overall better than MobileNet v1, MobileNet v2 and SqueezeNet, although its parameter number is almost twice larger than others. We have also observed similar patterns on our laptop and workstation, or on Jetson Nano with Pytorch, so this conclusion should at least have a certain degree of generalization. The justification could be that ResNet18 is the easiest one to leverage the parallel of computation.

Quantization Quantizing the model from FP32 to FP16 could slightly decrease the latency with almost no hurt to the performance in most of the cases. A corner case happened to the "sp_sparse" on high input resolution, where errors are dramatically larger on FP16 models. However, the model was provided by the repository author, and there could be some unknown details of the training process.

Input Resolution Reducing the input resolution is the easiest way to speed up SuperPoint inference, but that could also sacrifice the performance a lot, especially when it's downsized from medium to small. However, for edge devices like Jetson Nano, it's likely that the small resolution is a must if the inference speed is required to be larger than 10 FPS.

7 Future Work

Firstly, it'd be better if we can further train our SuperPoint models with more data, and evaluate the performance on more test sets. In this project, we only trained our models on MS-COCO2014, and tested the performance on KITTI. However, there're surely more options to try out in the future.

Secondly, we could fine-tune the detector and descriptor layers to increase both keypoint detection accuracy and speed. For example, we can expand the detector layers to have the same resolution as the input image to preserve the spatial information, and decrease the descriptor channel number to reduce the computation cost at the matching step. Besides, there's also a followup work, *UnSuperPoint* [11], with several improvements.

Furthermore, we can test replacing the matcher from the classical K-Nearest-Neighborhood method with SuperGlue [12], and potentially the overall performance could be better.

Lastly, it's worthwhile to optimize the cache locality and memory allocation of the neural network models. A more robust inference time means that in a real product, the scheduling of the computation resources will be much easier. For the memory allocation, the memory of edge devices is typically shared by CPU and GPU, but our code doesn't specially take that into account.

8 Ethical Considerations

Our project does not imply any potential ethical issue.

References

[1] D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superpoint: Self-supervised interest point detection and description," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2018, pp. 224–236.

[2] M. L. Researchers, "SuperPointPretrainedNetwork." [Online]. Available: https://github.com/magicleap/SuperPointPretrainedNetwork

[3] Y.-Y. Jau, "pytorch-superpoint." [Online]. Available: https://github.com/eric-yyjau/pytorch-superpoint

[4] Itseez, "Open source computer vision library," https://github.com/itseez/opencv, 2015.

[5] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision.* Ieee, 2011, pp. 2564–2571.

[6] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[7] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2015.

[8] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[9] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[11] P. H. Christiansen, M. F. Kragh, Y. Brodskiy, and H. Karstoft, "Unsuperpoint: End-to-end unsupervised interest point detector and descriptor," *arXiv preprint arXiv:1907.04011*, 2019.

[12] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, “Superglue: Learning feature matching with graph neural networks,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 4938–4947.

Appendices

A Graph Appendix

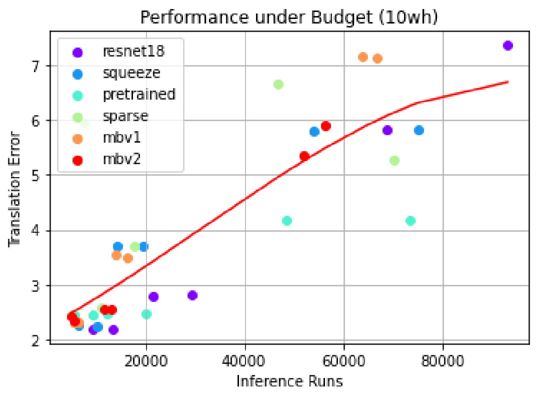


Figure 10: Performance (Translation Error) under Budget (10wh): Different Model Backbones

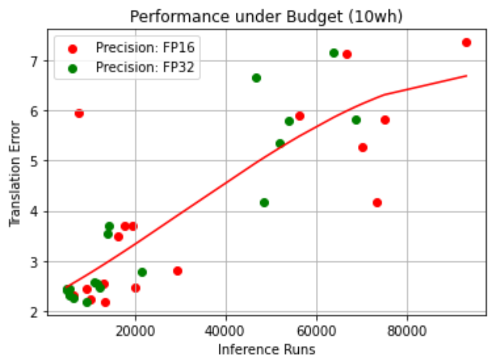


Figure 11: Performance (Translation Error) under Budget (10wh): Different Precisions

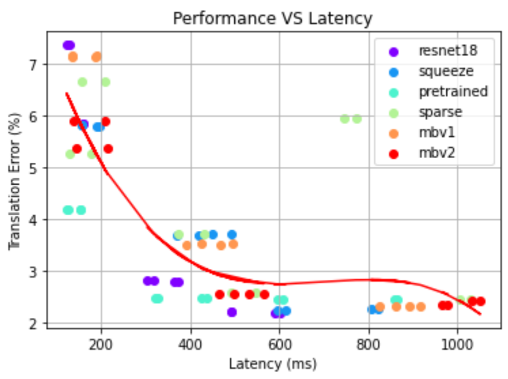


Figure 12: Latency VS Performance(Translation Error): Different Model Backbones

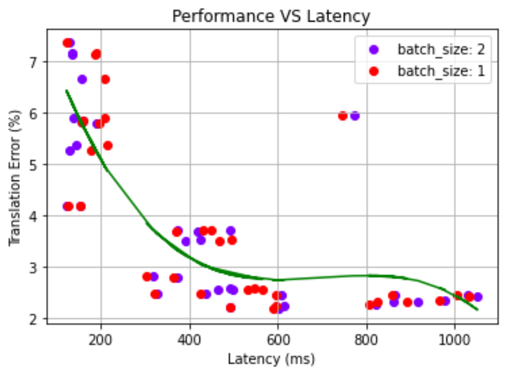


Figure 13: Latency VS Performance(Translation Error): Different Batch Size

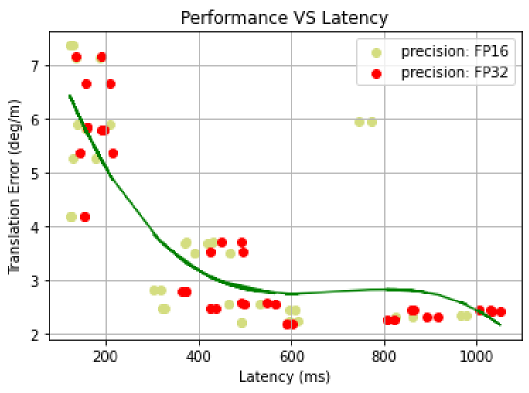
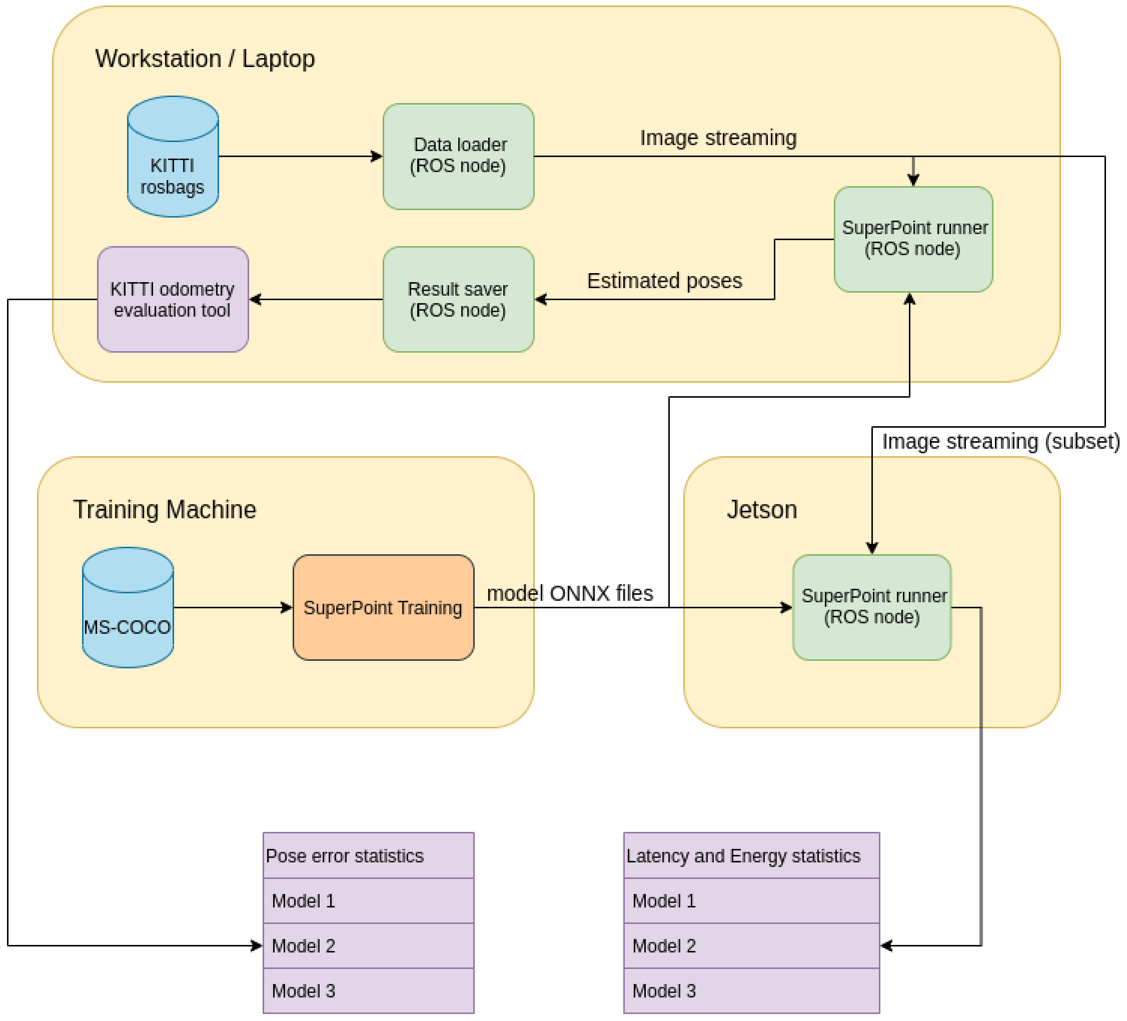


Figure 14: Latency VS Performance(Translation Error): Different Precisions

B System Diagram


Workstation / Laptop
KITTI rosbags
Data loader (ROS node)
Image streaming
SuperPoint runner (ROS node)
Estimated poses
KITTI odometry evaluation tool
Result saver (ROS node)
Image streaming (subset)
Training Machine
MS-COCO
SuperPoint Training
model ONNX files
Jetson
SuperPoint runner (ROS node)
Pose error statistics
Model 1
Model 2
Model 3
Latency and Energy statistics
Model 1
Model 2
Model 3