Robotics Final stage2

20191243 Jun Lee

⚠️Important Notice⚠️:

* Due to my poor computational resource, every time I tried to do localization and mapping on my UTM (M1 mac ubuntu VM) is stuck and terminated. I tried enormous of the effort to solve this but failed. So, I just used my local ros2-humble-mac, so it works quite well except for open3d package, which is not supported for the apple silicon. This is why I used the plyfile library instead of the open3d. So, you might need to install plyfile by command conda install -c conda-forge plyfile. only when if you want to save the map. I am very sorry about this uncomfortable decision. And I manually implemented the voxelization by numpy because I can’t use the Open3d Voxelization. So, please understand that the ply file may or may not be different from the type that generated from the open3d.

1. Analysis (bag play speed : 0.05 which is 20x slower than the normal)

텍스트, 스크린샷, 도표, 그래프이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 도표, 라인이(가) 표시된 사진

자동 생성된 설명

These are my results of the stage2 db3 files were applied by the algorithm that I’ve used in the stage1. As you can see here, At first, it is naturally APE, error is 0 because the origin is the initialized point. But as you can see here as time goes, the error increases proportionally. This is because the error of the target (map\_points) records cumulatively where the error of the real and estimate also cumulative. In this sense, we can expect the time proportionally increased error.

라인, 텍스트, 지도이(가) 표시된 사진

자동 생성된 설명지도, 라인, 텍스트이(가) 표시된 사진

자동 생성된 설명

This is my result from the ply viewer mac. Again, in my VM, it eventually terminate while ICP in run-time, and also my m1 mac local doesn’t support open3d conda environment, so I just attach this image by the local ply viewer.

예술, 물 튀김이(가) 표시된 사진

중간 신뢰도로 자동 생성된 설명

In here, as you can see many duplicated points are in the dense area. This might cause crucial error while doing ICP because it will eventually use the total size of the points of maps when finding correspondences. As we assumed the points with only one to one correspondence might tightly fit in the ICP, if not the majority can have duplicated target points which is not we wanted when we consider the speed and accuracy. Also, this is somehow noisy data. So that, we need to remove those of the noise points.

For the questions of the handout, I think it is not necessary to maintain all the points data. As I explained earlier, it maybe harm to the either speed or accuracy of the ICP, because of the duplicated target data. Also, maintaining enormous data (140MB for example) in all seconds are quite memory consuming. Instead, one thing we only need is just maintaining one point in such grid by the voxelization might help to improve those possible hazards.

1. Evaluation

* **Memory efficiency**: (The bag play speed is equally 0.05, 20x slower)
* ⚠️ (Note that my code is updated version for the optimization on the pose accuracy. So, the voxelized mapping will not show as the result below. You can save the map using the parameter self.map\_save = True, but in this case due to the lagging program in my computer the pose accuracy is not correct as the below estimation. So, the default I set is False)

[Suggestion] Voxelization until it can get the meaningful result in localization mapping.

Even though bag play file lagged in the baseline, so that the point cloud doesn’t perceive all of the data in the baseline, there were significant improvement of the memory size by the proposed method which perceives n times data then the baseline.

The base line, which is the exactly same code of the Stage 1’s memory consumption is following.

Only 73 times perceived of the all publishes.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

The first one is the old version which is base line, and the second one is the voxelized version with d = 0.1. Please note that the map\_best.ply file got the 399 published points which is significantly larger than the those of the baseline. With just a simple math we can calculate the difference of the ply storage in the same published points.

399 // 73 \* 194.1 // 24.4 = 39 this means that about 39x can reduce the memory size!

(Note that the voxel size is the appropriate one that it is compatible for the remaining metrics)

Then there might be one more question that what about voxel grid size up to get more down-sampled one? The answer is, there is no linear reduction due to the existence of the noise. Think about the lonely point in the space. Then no matter the voxel size is, the remaining number after down sampling steal same as 1. So, in this manner, if the points were not overlapped in the certain voxel grid, it might not be reduced so that It means the linear reduction is not expected.

In my calculation of the real data in the stage2, the voxel size 2x goes to the about 4 or 5x reducing in some certain period, which is in the significantly dense voxel d limitation. If not, the reduction is not as much as n times of the voxel size increased.

스케치, 그림, 예술, 일러스트레이션이(가) 표시된 사진

자동 생성된 설명 스케치, 그림, 그래픽, 일러스트레이션이(가) 표시된 사진

자동 생성된 설명

The left one is non voxelized and the right one is voxelized version.

This much improvement directly from the idea that pointing the representative points in each section. In real time case, as the points from the depth image is quite dense so that even if the correspondence points and the target points exactly match after the ICP, the result of the updating the map points will accept both. Which is truly memory consuming. Also, the ICP algorithm doesn’t match each point 100%, so that the correspondences will locate the nearby the target, but it may assume other points in terms of the map points. So, to prevent these problems, by representing one points as voxel, we can reduce the significant memory consumption.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Baseline | Voxel (d = 0.1) | Voxel (d = 0.3) |
| Storage[MB] | 194.1 | 24.4 | 12.1 |

* **Computation Speed**:
* (The bag play speed is equally 0.05, 20x slower, image is improve baseline order)
* [Suggestion] The y axis is fixed so that apply only x,z axis SVD using orthogonal projection.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

These are the computational time that includes the all of the following steps include compute\_icp().

텍스트, 폰트, 스크린샷, 흑백이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 화이트, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

These are the computational time that only includes the compute\_icp().

The overall improvement is like net, so the memory consumption decrease affects the speed of the computation. Because of the managing memory significantly decreased, the overall speed increased by reducing I/O burden. But what about the speed except the managing the memory?

The second two images demonstrate this. The overall speed of the new version of the localization and mapping remains the same as 0.12 but the old version of the ICP takes about 20 seconds which is about 17x slower than the improved version.

Someone might argue that it might be the reason of the target and source size difference by the voxelization, so that I fixed the size of source and target as 500 and again computed only the one iteration at the SVD and these are the results. **Which delete the effect of the voxelization. And below is the result.**

텍스트, 폰트, 스크린샷, 흑백이(가) 표시된 사진

자동 생성된 설명 텍스트, 폰트, 스크린샷, 흑백이(가) 표시된 사진

자동 생성된 설명

The left one I baseline and the right one is my suggestion. As you can see here the lowest time is quite similar but the the baseline version which use the 3x3 matrix, the highest time consumed is 0.0062 around 1.8x slower than my suggestion. This is because the reduction of the degree of freedom of the matrix to compute in ICP-SVD.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Overall** | **Compute\_ICP()** | **SVD**  **(same target and source)**  **Which delete the effect of the voxelization** |
| Baseline(s) | 0.17 | 0.13 | 0.00034 |
| Suggest(s) | 33.7 | 18.8 | 0.00047 |
| Bag speed | 0.05(20x slower) because I want to see full computation without bothering | | |

As you can see above, the algorithm of the voxelization make amazing time reduction. But as you can see here the computing ICP by SVD in the same condition where the computing number of target and source points are same, the improvement seems quite good because it keeps doing this for 300times in for loop. This small improvement also very important in terms of the overall speed reduction.

* **Pose accuracy**:
* (The bag play speed for my updated code is equally 1 normal speed)
* ⚠️Important Notice⚠️: I don’t know why, but in my case without the align parameter goes well, but in the case of the align it seems awkward. Maybe caused by the axis setting, but anyhow, I did the command by following:

*evo\_ape tum /Users/leejun/Downloads/my\_pkg/result/stage2\_gt\_pose.txt /Users/leejun/Downloads/my\_pkg/my\_pkg/odometry\_data.txt -p*

As far as I know, *without alignment* might be more difficult and my code did well in this bad situation.

Please set self.map\_save = False when you evaluate the accuracy.

* [Suggestion] The y axis is fixed so that apply only x,z axis SVD using orthogonal projection(2d). And also, the added data to the target (map points) is only the data which cannot find the correspondence. Let me say this strategy as **golden data strategy**. Which means the source data which the distance between the correspondence exceeds the threshold. With that, I applied the updating threshold algorithm from the **KissICP**.
* 텍스트, 소프트웨어, 멀티미디어 소프트웨어, 그래픽 소프트웨어이(가) 표시된 사진

  자동 생성된 설명
* Like this image, the only updating target points are missed correspondence points from source point. Which is the outlier. Then we can get abstract and accurate map as the image represents. (Please understand map will seems to be weird, because I only concentrate to the pose accuracy and speed. If you add the save points function, then it will show the whole point cloud but somehow slower and accuracy can decrease by the absence of the interpolate frame.)

텍스트, 스크린샷, 도표, 그래프이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 도표, 라인이(가) 표시된 사진

자동 생성된 설명

텍스트, 영수증, 폰트, 화이트이(가) 표시된 사진

자동 생성된 설명

These are my **baseline** best result which performs the best among the baseline test by manually changing the bag speed. And this is the speed of 0.05 which is 20x slower than the normal.

But in my algorithm, it seems very accurate when compare by the baseline even if the bag play running speed is 20x faster than the baseline, which is just normal speed.

텍스트, 도표, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 그래프, 도표이(가) 표시된 사진

자동 생성된 설명

텍스트, 영수증, 폰트, 화이트이(가) 표시된 사진

자동 생성된 설명

This is the result before applying the **golden data.** Which means that it only applies the transformation of the xz plane only. As you can see here, the accuracy is not that good because if the data that is added to the map points, if it is not accurate data, the error is cumulatively increased.

텍스트, 스크린샷, 그래프, 도표이(가) 표시된 사진

자동 생성된 설명텍스트, 도표, 스크린샷, 그래프이(가) 표시된 사진

자동 생성된 설명

텍스트, 영수증, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

This is the result of without alignment when apply evo-ape and after applying **golden data strategy**. Which is the final code. As you can see this image, the overall accuracy is very high.

This good improvement directly come from the making golden data. Most of the errors come from the complex added data which is having an error. But if we make the golden data which is not likely to change because it treated as the default. So, by doing this, the first captured space, which is likely to most accurate data among all received data. (You can imagine the accuracy of the first few depth map from the bag play.) In ICP will be treated these as default so there might not be any confusion by the false data.

지도이(가) 표시된 사진

자동 생성된 설명

This is the version that includes the map, but here as the I/O time consumes, the buffer by the bag play is not going well as the self.map\_save = False version, which is the default, because of the lacking behind computation compared by self.map\_save = False which doesn’t require further I/O operation and voxelization. So, the result here as you can see, it is worse than the optimized version above.

|  |  |  |  |
| --- | --- | --- | --- |
|  | baseline | Only XZ plane projection | XZ plane projection, golden data strategy  (Final) |
| Mean Error (m) | 6.72 | 7.43 | 1.89 |
| RMSE | 9.52 | 9.3 | 2.11 |
| Bag play rate | 0.05 | 1 | 1 |

The reason of the accuracy improvement is that we set the movement in only XZ plane which reduces the computation amount and increase the speed and accuracy as well by reducing the Y axis computation. But the better improvement comes from the golden data strategy, which saves the outlier of the source points (where np.array(distances) > self.thres), because it reduces duplicated and cumulative error by permitting only one point as true value among multiple points from each receiving point clouds. This is similar with the case of the ML. If you give one input and multiple different outputs, then the ML model cannot no which output to choose. But by my golden data strategy, we can give them only one corresponding output and then it is easy and accurate to calculate. So, in these reasons, I can get better result.