Localization of autonomous robot using 3D grid puzzle map for optimizing computation resources

Jaeguk Byeon[1], Eugene Kim[2], Myeonghwan Hwang[2], Seungha Yoon[2], and Hyunrok Cha[2,*]

***Abstract*— The autonomous robot has evolved with the advancement of AI (Artificial Intelligence) technologies, leading to significant improvements in performance. However, these advancements have increased computational demands, requiring higher processing ability and real-time capabilities. In particular, when autonomous robot is performed in a large-scale environment where GPS is unreliable, the computational burden of localization tasks continues to grow. To overcome this challenge, the submap strategy has been introduced, which divides a global map generated through Simultaneous Localization and Mapping into smaller submaps. The submap approach incorporates overlapping regions between submaps to ensure smooth localization during transitions. On the other hand, the submap method also increases storage requirements due to redundant map regions. Moreover, using fixed-size submaps for localization may include unnecessary environmental information, reducing efficiency. This study presents a method that inherently does not allow overlapping regions between submaps, reducing storage requirements while ensuring continuous environmental information during transitions. Furthermore, we propose an approach that dynamically adjusts the provided map area based on the movement direction of agents, improving the efficiency of localization.**

I. INTRODUCTION

The advancements in sensors such as LiDAR (Light Detection And Ranging), IMU (Inertial Measurement Unit), and Camera, along with AI technologies, have significantly contributed to the development of autonomous robot technology [1], [2]. These advances have improved the performance of autonomous robot technologies. However, it has also increased the computational burden by requiring real-time processing of heavy computation [3]. This issue also presents in localization tasks within large-scale environments where GPS is unreliable.

As the operational area of an autonomous robot agents expands, the size of the map also increases, leading to a rise in both computational complexity and memory usage during high-precision localization task using 3D dense map acquired from LiDAR [4].

As a countermeasure, the submap method which divides large-scale 3D dense map is emerged [5]. A traditional submap method generates submap files by segmenting the global map with overlapping regions to provide continuous region information during submap transitions. However, the presence of overlapping regions increases the stored file size. Additionally, when fixed-size submap files are utilized, unnecessary regional information may be included regarding the trajectory of agents.

[1]Author is with Robot Engineering, Korea National University of Science and Technology, Daejeon(postal code : 34113), South Korea wornr7390@kitech.re.kr

[2]Authors are with the Purpose-Based Mobility group, Seonam Division, Korea Institute of Industrial Technology, Gwangju 61012, South Korea

*Corresponding authoor, hrcha@kitech.re.kr

A. Contributions

In this study, we propose a method for generating submaps without overlapping regions to reduce storage size while providing continuity and valid region information in the direction of the agent's movement. Fig. 1 shows a comparison between the localization process using the traditional submap method and the localization process using the our proposed method(grid puzzle map). In traditional submap generation method, overlapping regions are created between each submap to provide the agents with continuous area information. Leading to an increase in the file size of each submap proportional to the overlapping sections. On the other hand, in our proposed method, the global map is divided into small grid-shaped segments like puzzle pieces, using the origin of the global coordinate system as a reference. Based on the position of agents, the matching grid puzzle map files within a certain range are retrieved. The loaded files are then assembled like puzzle pieces, aligning them with their original positions on the global map. In this process, the number of loaded grid puzzle map files is dynamically adjusted based on the movement direction of agents to provide continuous spatial information of the surrounding area for the localization algorithm. The proposed approach removes overlapping regions between grid puzzle map files, reducing storage memory usage.

II. METHODOLOGY

A. Generate grid puzzle map

The point cloud map file generated through the 3D LiDAR sensor-based mapping process are divided along the x-axis and y-axis based on origin of the map. The global map is divided into a grid with equal side lengths, and each segmented region along the x-axis and y-axis includes all z-axis point cloud samples within its boundaries. To load the grid puzzle maps of the surrounding area based on the position of agents, each grid puzzle map file is saved using the x and y coordinates of its region center as the file name. Let $\mathbf{P} = \{\mathbf{p}_i \mid \mathbf{p}_i \in \mathbb{R}^3,\ i = 1, 2, \ldots, N\}$ be the set of global map points, where $\mathbf{p}_i = (x_i, y_i, z_i)$ represents the coordinates of the i-th point on the global map. The $\mathbf{x}$ and $\mathbf{y}$ center coordinates $\mathbf{c}_{\mathbf{x}i}$, $\mathbf{c}_{\mathbf{y}i}$ of the grid boundary to which each point $\mathbf{p}_i$ belongs are computed as follows:

This is a preprint version.The final authenticated publication is available at https://doi.org/10.1109/ARSO64737.2025.11124984

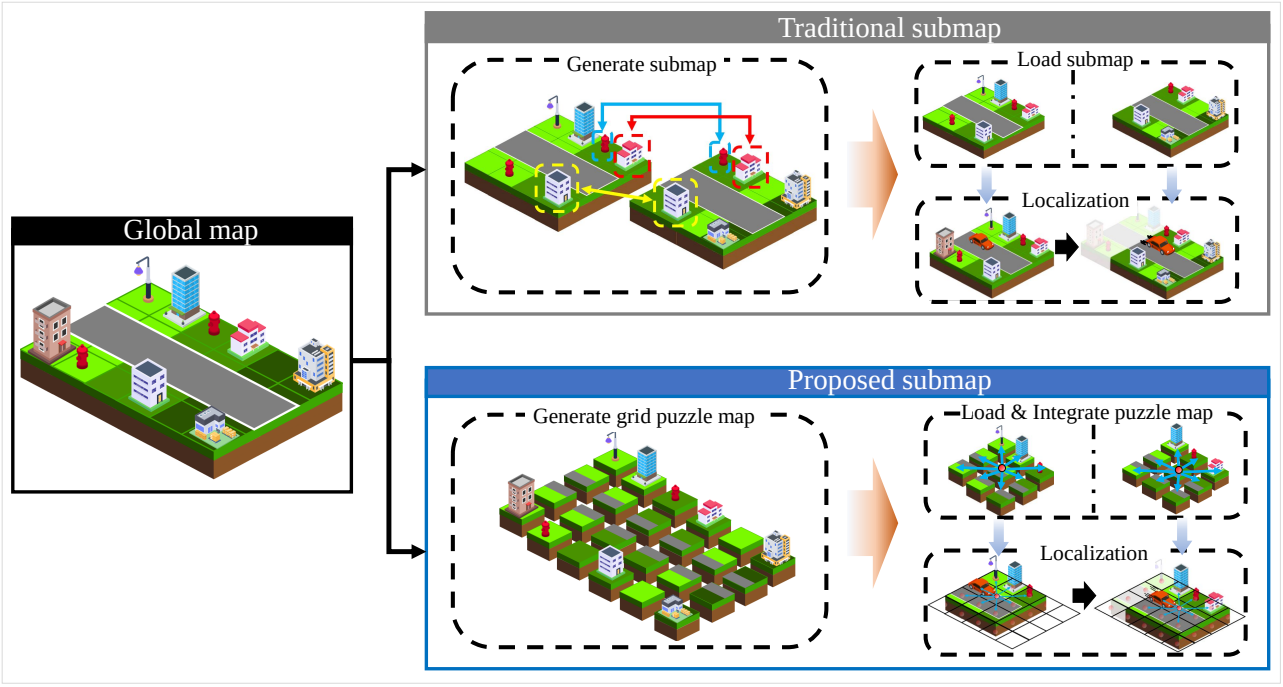

Fig. 1. The diagram illustrates the traditional submap method and the proposed submap method for global map files. The traditional submap method generates fixed-size divided map files, including overlapping regions between submaps. During localization, the fixed-size submap files are loaded and utilized. On the other hand, in the proposed submap method, the global map is divided into small grid-shaped segments. The divided map files do not have overlapping regions and are stored with their center coordinates as file names. During localization, the required grid map files are dynamically loaded and integrated for use.

$$\begin{aligned} \mathbf{c}_{\mathbf{x}i} &= \left\lfloor \frac{\mathbf{x}_i}{s} \right\rfloor \cdot s + \frac{s}{2}, \\ \mathbf{c}_{\mathbf{y}i} &= \left\lfloor \frac{\mathbf{y}_i}{s} \right\rfloor \cdot s + \frac{s}{2}, \end{aligned} \tag{1}$$

where s is the length of one side of the grid used for region segmentation, $\lfloor \cdot \rfloor$ is the floor function rounds down the given value to the nearest integer multiple of s. These procedures are shown in Algorithm 1.

B. Using grid puzzle map for localization

When the coordinates of the agents are a_x, a_y , the center coordinates of the nearest grid region surrounding the agents $\mathbf{c}'_{\mathbf{x}}$, $\mathbf{c}'_{\mathbf{y}}$ are calculated as follows:

$$\begin{aligned} \mathbf{c}'_{\mathbf{x}} &= \left\lfloor \frac{a_x}{s} \right\rfloor \cdot s + \frac{s}{2}, \\ \mathbf{c}'_{\mathbf{y}} &= \left\lfloor \frac{a_y}{s} \right\rfloor \cdot s + \frac{s}{2}. \end{aligned} \tag{2}$$

The grid puzzle map file corresponding to $\mathbf{c}'_{\mathbf{x}}$ and $\mathbf{c}'_{\mathbf{y}}$ is selected based on its filename. Subsequently, adjacent grid puzzle map files are also retrieved. Because all grid puzzle map files are segmented into uniformly sized grids, their filenames, representing the central coordinates, maintain a consistent offset interval. Using this property, the neighboring map files are selected as follows:

$$\mathbf{C} = \sum_{u=u_{\min}}^{u_{\max}} \sum_{v=v_{\min}}^{v_{\max}} \left(\mathbf{c}'_{\mathbf{x}} + u \cdot s,\ \mathbf{c}'_{\mathbf{y}} + v \cdot s \right), \tag{3}$$

Algorithm 1: Generation of Grid Puzzle Map

Input: Global point cloud $\mathcal{P} = \{ \mathbf{p}_i = (x_i, y_i, z_i) \}_{i=1}^{N}$, grid side length s

Output: Set of grid puzzle map files indexed by their center coordinates

Function *ComputeBuckets*$(\mathcal{P}, s)$
- initialize empty dictionary `buckets`;
- **foreach** $\mathbf{p}_i \in \mathcal{P}$ **do**
  - // compute grid indices and center (see Eq. 1);
  - $i \leftarrow \lfloor x_i / s \rfloor, \quad j \leftarrow \lfloor y_i / s \rfloor$;
  - $c_x \leftarrow i \cdot s + \frac{s}{2}, \quad c_y \leftarrow j \cdot s + \frac{s}{2}$;
  - append $\mathbf{p}_i$ to `buckets`$[(i, j)]$;
- **return** *buckets*;

1) partition all points into grid buckets;
`buckets` $\leftarrow$ ComputeBuckets$(\mathcal{P}, s)$;

2) save each bucket as a separate map file;
foreach *key* (i, j) *in* *buckets* **do**
- let (c_x, c_y) be $\left(i \cdot s + \frac{s}{2},\ j \cdot s + \frac{s}{2} \right)$;
- filename $\leftarrow$ c_x _c_y `.pcd`;
- save point cloud `buckets`$[(i, j)]$ to `filename`;

return *all generated grid puzzle map files*;

This is a preprint version.The final authenticated publication is available at https://doi.org/10.1109/ARSO64737.2025.11124984

where $\mathbf{C}$ is the set of neighbor grid center coordinates based on $(\mathbf{c'_x}, \mathbf{c'_y})$, $u,\ v$ are offsets for grid expansion ($u,\ v \in \mathbb{Z}$, $\mathbb{Z}$ is the set of all integers), and $u_{\min},\ u_{\max},\ v_{\min},\ v_{\max}$ are the range of u and v. These ranges also represent the search ranges for map files.

When the agents move beyond a certain distance from $(a_x,\ a_y)$, its current position is updated, and the neighbor grid puzzle map files are searched using equations (2), (3) with updated position. When the newly updated coordinates of the agents and the previously used coordinates of agents are respectively $(a_x^{\text{curr}},\ a_y^{\text{curr}})$, $(a_x^{\text{prev}},\ a_y^{\text{prev}})$, the movement vector of agents $\vec{a}$ and angle θ are obtained as follows:

$$\begin{aligned} \vec{a} &= (a_x^{\text{curr}},\ a_y^{\text{curr}}) - (a_x^{\text{prev}},\ a_y^{\text{prev}}) \\ \theta &= \cos^{-1}\left(\frac{\vec{a}\cdot\vec{e}_x}{\|\vec{a}\|\|\vec{e}_x\|}\right) \end{aligned} \tag{4}$$

where the angle θ between the movement vector of agents $\vec{a}$ and x-axis unit vector in the global coordinate system $\vec{e}_x$ is calculated using the dot product formula.

The range of u and v in equation(3), determined using the θ, as follows:

$$\begin{aligned} u_{\min} &= n + \Delta u_{\min}(\theta), \quad u_{\max} = n + \Delta u_{\max}(\theta), \\ v_{\min} &= -n + \Delta v_{\min}(\theta), \quad v_{\max} = -n + \Delta v_{\max}(\theta), \end{aligned} \tag{5}$$

where $n \in \mathbb{Z}$, and n used to define the default search range of neighboring map files as a square region of $(2n+1) \times (2n+1)$. Then θ is utilized to adjust the search range by considering the movement direction of agents.The additional search range of map files based on the θ ranges is presented in Table I.

TABLE I
SEARCH RANGE ADJUSTMENT BASED ON ANGLE θ

θ Range	$(\Delta u_{\min}, \Delta u_{\max})$	$(\Delta v_{\min}, \Delta v_{\max})$
$-20° \le \theta < 20°$	$(0, k)$	$(0, 0)$
$20° \le \theta < 70°$	$(0, k)$	$(0, k)$
$70° \le \theta < 110°$	$(0, 0)$	$(0, k)$
$110° \le \theta < 160°$	$(-k, 0)$	$(0, k)$
$160° \le \theta < 200°$	$(-k, 0)$	$(0, 0)$
$200° \le \theta < 250°$	$(-k, 0)$	$(-k, 0)$
$250° \le \theta < 290°$	$(0, 0)$	$(-k, 0)$
$290° \le \theta < 340°$	$(0, k)$	$(-k, 0)$

Fig. 2 shows the map file search range based on the movement direction of agents,the range of θ. The loaded map files are integrated into a single dataset and provided as the map file dataset for the localization algorithm. The proposed method removes the overlapping regions between submaps when dividing the global map, reducing the stored file size. The proposed approach also improves efficiency in dynamically loading the required map files based on the movement direction of agents. This approach, similar to conventional submap-based methods, is expected to reduce memory usage and CPU utilization while performing localization. These procedures are shown in Algorithm 2.

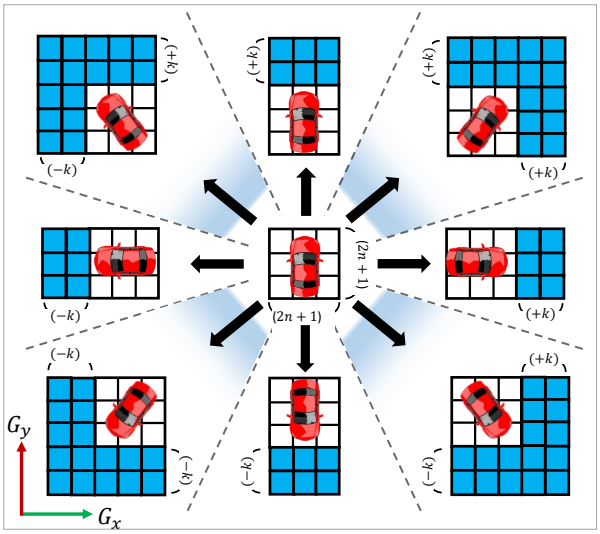

Fig. 2. The submap search range based on the movement direction vector of agents and the range of the θ with respect to the x-axis ($\mathbf{G}_x$) and y-axis ($\mathbf{G}_y$) of the global coordinate system. The search range is initially a square of size (2n+1) x (2n+1). The movement direction vector of agents, represented by a black arrow, shows the extended search area based on the θ range, which is separated by a dashed line. The additional search range corresponding to the movement direction of agents is indicated by a blue rectangle.

Algorithm 2: Localization via Dynamic Grid Puzzle Map Loading

Input: Current pose (a_x, a_y), previous pose $(a_x^{\text{prev}}, a_y^{\text{prev}})$, grid size s, default search radius n, expansion factor k

Output: Integrated point cloud $\mathcal{M}$ for localization

1. Compute nearest grid center to current pose (Eq. 2);
$c'_x \leftarrow \lfloor a_x/s \rfloor\, s + \frac{s}{2}, \quad c'_y \leftarrow \lfloor a_y/s \rfloor\, s + \frac{s}{2}$;

2. Compute movement vector and heading angle (Eq. 4);
$\vec{a} \leftarrow (a_x, a_y) - (a_x^{\text{prev}}, a_y^{\text{prev}})$;
$\theta \leftarrow \cos^{-1}\left(\frac{\vec{a}\cdot\mathbf{e}_x}{\|\vec{a}\|\|\mathbf{e}_x\|}\right)$;

3. Determine neighbor offsets based on θ (Eq. 3 & Table I);
Compute $(u_{\min}, u_{\max}, v_{\min}, v_{\max})$ from n, k, and θ;

4. Load and merge grid files within the computed window;
$\mathcal{M} \leftarrow \emptyset$;
for $u \leftarrow u_{\min}$ **to** $u_{\max}$ **do**
  for $v \leftarrow v_{\min}$ **to** $v_{\max}$ **do**
    $c \leftarrow (\, c'_x + u\, s,\ c'_y + v\, s\,)$;
    filename $\leftarrow c'_x + u\, s\_c'_y + v\, s$`.pcd`;
    load point cloud from `filename` into $\mathcal{M}$;

5. Optional: trigger re-computation only if $\|\vec{a}\| > s/2$ to avoid redundant loads;
return $\mathcal{M}$;

III. Experiment

A. General setup

The experimental setup utilized the ROS(Robot Operating System) Noetic framework running on the Ubuntu 20.04 operating system. The CPU model used is the AMD Ryzen 7 9800X3D 8-core Processor. In this experiment, we used NDT (Normal Distributions Transform) localization, one of the prior-map based localization methods. The localization computation was performed using a multi-threaded approach with the **ndt omp**[1] library.

B. Experiment conditions

The experiments were conducted using the courtyard and urban environment datasets. Before performing localization for each dataset, the global map was pre-generated using the Fast LIO2 [6] algorithm. The grid size s was set to 2 m, and grid puzzle map files were generated for each dataset global map. Afterward, localization was performed on the two datasets, collecting statistics on CPU utilization and memory usage for scenarios using the global map and those using the grid puzzle map. The CPU utilization and memory usage statistics were obtained using the **psutil** library in Python. In the scenario utilizing the global map, the statistics were collected only for the process performing NDT localization. In the scenario using the grid puzzle map, the statistics were gathered for both the NDT localization process and the process of providing the grid puzzle map, and the final results were obtained by summing the values. The localization accuracy of the grid puzzle map was evaluated by comparing it with the global map. We calculated the error between the two trajectories obtained using each method, based on their respective path coordinates, using the Root Mean Square Error (RMSE).

C. Dataset scenario

1) Courtyard dataset scenario: The map file has a planar area of approximately (150 m × 200 m), and the trajectory length is about 0.646 km. Localization was performed for a total duration of 1,269 seconds by walking. The experiment was conducted using ndt omp with two CPU cores. The values of n and k in equation (3) and Table I were set respectively to 17 and 10. The Lidar sensor used in the experiment was the Livox-Mid360 model.

2) Urban dataset scenario: We used UrbanNav-HK-Medium-Urban-1 Dataset from IPNL-POLYU[2]. The map file has a planar area of approximately (480 m × 710 m), and the trajectory length is about 3.64 km. Localization was performed for a total duration of 785 seconds by driving vehicle. The experiment was conducted using ndt omp with four CPU cores. The values of n and k in equation (3) and Table I were set respectively to 25 and 20. The Lidar sensor used in the experiment was the Velodyne HDL-32E model.

[1]https://github.com/koide3/ndt$_o mp$

[2]https://github.com/IPNL-POLYU/UrbanNavDataset

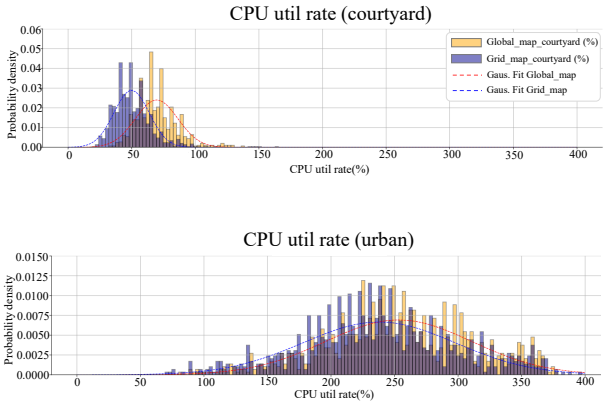

Fig. 3. These result graphs compares the CPU utilization rate in the localization scenario of the Courtyard (above figure) and Urban (below figure) datasets, with and without the application of the proposed method. When the proposed method was applied, the average CPU utilization rate decreased compared to when it was not applied.

IV. Results and Discussions

A. Comparison of CPU utilization rate

Fig. 3 shows the CPU utilization when using the global map and the grid puzzle map for each dataset. In the courtyard dataset, the average CPU utilization when using the global map was 74.79%. When the proposed method was applied, the average CPU utilization was 56.06%. It showed a reduction of 18.73%. In the urban dataset, the average CPU utilization was 253.28% with the global map and 237.89% with the proposed method, resulting in a 15.39% reduction.

B. Comparison of memory usage

Fig. 4 shows the memory usage for each dataset with and without the proposed method. In the courtyard dataset, the average memory usage without applying the proposed method was 166.01 MB, whereas with the proposed method, it was 111.02 MB, resulting in a 33.12% reduction. In the urban dataset, the average memory usage without the proposed method was 509.38 MB and the average memory usage with the proposed method 193.12 MB, resulting in a 62.09% reduction.

C. Comparison of localization

Fig. 5 shows a comparison of the localization trajectories in the courtyard dataset when using the global map and when applying the proposed method. The RMSE of the trajectory error between the two approaches was 0.0433 m. Fig. 6 shows the comparison of the localization trajectories in the urban dataset with and without the proposed method. The RMSE of the trajectory error was 0.0677 m.

D. Comparison of performance with and without the proposed method

From the result, the proposed method effectively reduces CPU utilization and memory usage compared to using the global map. For the courtyard and urban dataset scenarios, the size of the generated global map files was 631.1MB and 1,267.4MB, respectively. Using the proposed method, the

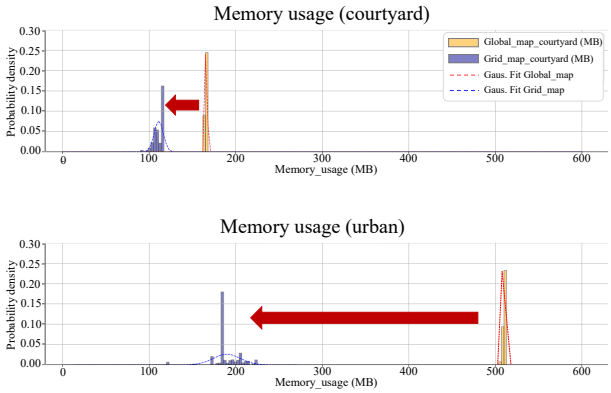

Fig. 4. These result graphs compares the memory usage in the localization scenario of the Courtyard (above figure) and Urban (below figure) datasets, with and without the application of the proposed method. When the proposed method was applied, the average memory usage decreased compared to when it was not applied.

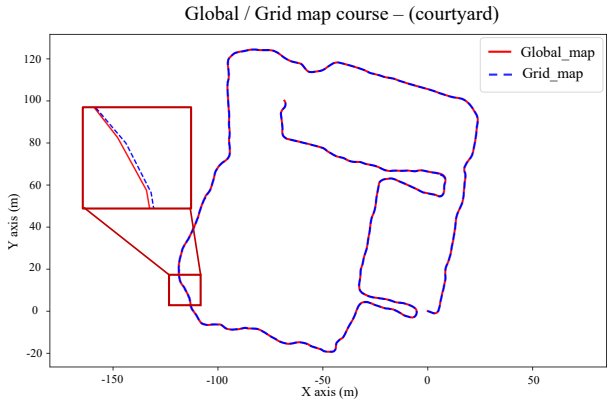

Fig. 5. The trajectory of agent comparing localization using a global map and localization using the proposed method in the courtyard dataset.

number of divided map files was 8,282 and 47,480, while the total file size remained identical to that of the global map files. This result demonstrates that the proposed method generates submap files without requiring extra storage due to overlapping regions. Furthermore, a comparison of the trajectories obtained using the global map and the proposed method showed no significant differences. This indicates that the proposed method provides continuous region information and effective range grid puzzle map files.

V. CONCLUSIONS

The submap strategy emerged as a solution to alleviate the computational burden of localization tasks in large-scale environment. However, the traditional submap strategy generates submap files with overlapping regions, which increases storage requirements compared to a single global map file. Additionally, some unnecessary regional information may be included, regardless of the movement direction of the autonomous robot agents because traditional approach uses fixed-size submap. In this study, we proposed the method that divides the global map file into small units like puzzle pieces and utilizes grid puzzle map files corresponding to

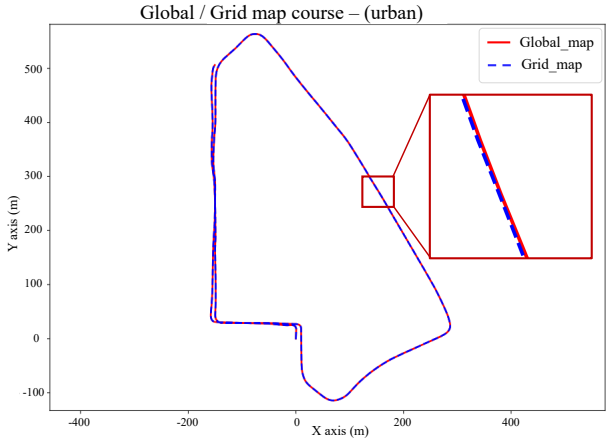

Fig. 6. The trajectory of agent comparing localization using a global map and localization using the proposed method in the urban dataset.

the location of agents. The proposed method achieved the primary goal of the submap strategy by reducing CPU utilization and memory usage compared to using a single large-scale global map file. The results confirmed that proposed method generates submap files without overlapping regions between submap files, resulting in lower storage requirements compared to the traditional submap generation method. The divided map files using the proposed method were same total size as a single global map file, without additional storage consumption. Additionally, we applied a method that dynamically adjusts the range of provided region information based on the movement direction of agents.

As future work, it is necessary to conduct research on applying a neural network model instead of simply using the movement vector of agents to determine its direction, in order to provide a more accurate and optimized range of regional information.

ACKNOWLEDGMENT

This work was supported by ”Development of Core Technologies for a Working Partner Robot in the Manufacturing Field” (KITECH EO-250005).

REFERENCES

[1] D. J. Yeong, G. Velasco-Hernandez, J. Barry, and J. Walsh, “Sensor and sensor fusion technology in autonomous vehicles: A review,” *Sensors*, vol. 21, no. 6, p. 2140, 2021.

[2] J. Fayyad, M. A. Jaradat, D. Gruyer, and H. Najjaran, “Deep learning sensor fusion for autonomous vehicle perception and localization: A review,” *Sensors*, vol. 20, no. 15, p. 4220, 2020.

[3] X. Wang, M. A. Maleki, M. W. Azhar, and P. Trancoso, “Moving forward: A review of autonomous driving software and hardware systems,” *arXiv preprint arXiv:2411.10291*, 2024.

[4] E. Kruzhkov, A. Savinykh, P. Karpyshev, M. Kurenkov, E. Yudin, A. Potapov, and D. Tsetserukou, “Meslam: Memory efficient slam based on neural fields,” in *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 430–435, IEEE, 2022.

[5] Y. Feng, Z. Jiang, Y. Shi, Y. Feng, X. Chen, H. Zhao, and G. Zhou, “Block-map-based localization in large-scale environment,” in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1709–1715, IEEE, 2024.

[6] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, “Fast-lio2: Fast direct lidar-inertial odometry,” *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2053–2073, 2022.