УДК 004.94

БІГАНСЬКИЙ Б. М.*, КОВАЛЮК Д. О.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

ПОРІВНЯННЯ ПРОГРАМНИХ РЕАЛІЗАЦІЙ АЛГОРИТМУ BUNDLE ADJUSTMENT ДЛЯ ЗАДАЧІ SLAM

В статті розглянуто задачу SLAM, зокрема один з головних її етапів Bundle Adjustment – відновлення тривимірної структури сцени та параметрів камери. Наведено існуючі бібліотеки, що реалізують даний алгоритм. Запропоновано програмне рішення для порівняння бібліотек Bundle Adjustment. В результаті досліджено швидкодію, кількість ітерацій, абсолютну помилку позиції (APE) та значення помилки. Здійснено візуалізацію розв'язку задачі Bundle Adjustment. Проведено порівняльний аналіз ефективності бібліотек Ceres, GTSAM та SymForce на наборах даних Ladybug, Trafalgar Square, Dubrovnik. Результати експериментів показали, що SymForce забезпечує найкращий баланс між швидкістю, точністю та збіжністю, тоді як Ceres демонструє високу швидкодію для окремих сценаріїв, а GTSAM поступається за всіма критеріями. Отримані висновки є важливими у виборі оптимального інструменту для розв'язання задач SLAM та нелінійної оптимізації.

Ключові слова: *SLAM, Bundle Adjustment, оптимізація, програмне забезпечення, комп'ютерний зір, робототехніка*

DOI: 10.20535/2617-9741.1.2025.325837

* Corresponding author: bbm-ihf@lll.kpi.ua
Received 27 December 2024; Accepted 22 January 2025

Постановка проблеми. Сьогодні робототехніка знайшла широке застосування в різних галузях, і успішно замінює ручну працю у важких або небезпечних умовах. Однією із задач для функціонування роботів є визначення їх поточного місця знаходження – локалізація. Загалом існує два підходи до локалізації [1]:

1) Mapping then Localizing. Спочатку створюється карта середовища, потім робот використовує цю карту для визначення свого місця. Цей метод потребує менше обчислювальних ресурсів, але передбачає наявність попередньо створеної карти.

2) SLAM (Simultaneous Localization and Mapping). Робот самостійно під час руху будує карту навколишнього середовища (mapping) та визначає своє положення в режимі реального часу (localization).

Підхід SLAM є більш складним у розрахунках, але дозволяє працювати в середовищах без попередньої карти. Саме тому він може використовується у промисловості (3D-картографування, інспекція та обслуговування заводських приміщень), військовій сфері (автономні бойові дрони і наземні машини, що мають орієнтуватися на місцевості без GPS), логістиці (складські роботи-маніпулятори та автономні візки AGV, що працюють в динамічному середовищі), сільське господарство (моніторинг полів).

Математичний опис задачі SLAM можна представити наступним чином [2]: в якості вхідних даних маємо послідовність зображень (кадрів) середовища, отриманих камерою або набором камер у моменти часу від $t = 0$ до $t = n$.

$$I = \{I_0, I_1, I_2, \dots, I_n\} \tag{1}$$

де I – окремий кадр.

В якості вихідних даних є траєкторія камер (робота) та карта середовища. Траєкторія - це набір поз у просторі:

$$X = \{x_1, x_2, x_3, \dots, x_\tau\} \tag{2}$$

де x_τ – це поточна поза камери в момент часу τ (зазвичай виражається як матриця обертання R та зсув відносно центру координат (трансляція) який позначається t).

Карта середовища – набір 3D-точок, що репрезентують структуру простору:

$$M = \{p_1, p_2, p_3, \dots, p_\tau\} \tag{3}$$

де p_τ – це 3D-координати точок у просторі.

Основна мета SLAM полягає в тому, щоб маючи вхідні зображення I, знайти траєкторію камери X та карту середовища M, максимізуючи апостеріорну ймовірність цих змінних. Це означає, що потрібно знайти такі X і M, які найкраще пояснюють отримані кадри I.

Зазначимо, роботизовані комплекси характеризуються різними апаратними можливостями:

– дрони та мобільні роботи працюють в умовах обмеженої ваги та енергоспоживання, тому їм потрібні легкі та швидкі оптимізаційні алгоритми;
– автономні транспортні засоби та складські роботи можуть мати більше обчислювальних ресурсів, але працюють в динамічному середовищі, де важлива швидкість обчислень;
– роботи для інспекції та картографування можуть дозволити собі більш ресурсномісткі методи, але їм важлива висока точність.

Враховуючи обмежені обчислювальні потужності технічних засобів, що використовуються в робототехніці (наприклад, Raspberry Pi, Nvidia Jetson, тощо), постає задача дослідження ефективності програмної реалізації алгоритмів SLAM та створення рекомендацій для їх використання.

Аналіз попередніх досліджень. У SLAM багато років домінували фільтраційні методи, але зараз вони поступово витіснилися підходами, заснованими на оптимізації [3]. Підхід з оптимізацією був вперше представлений в [1], але спочатку не набув популярності через високу обчислювальну складність. Сьогодні, завдяки зростанню обчислювальних потужностей, методи з оптимізацією стали передовим стандартом і дозволяють швидко та точно розв'язувати задачі оцінювання в SLAM.

Загальна схема роботи на основі оптимізації, відповідно до [4], представлена на рисунку 1:

Рис. 1 – Загальна схема роботи SLAM на основі оптимізації

SLAM на основі оптимізації можна розділити на 2 великих частини:

– Front-end алгоритми – відповідають за оцінку положення камери та побудову 3D карти;
– Back-end алгоритми – відповідають за оптимізацію оцінених положень камери та карти.

Front-end алгоритми, як правило включають наступні етапи: виділення ключових точок (Feature Extraction), зіставлення точок (Feature Matching), оцінка положення камери (Pose Estimation), триангуляція – знаючи положення камери обчислення 3D-координати точок середовища (Triangulation), локальне створення карти (Local Mapping).

До Back-end алгоритмів (оптимізація) включають наступні:

– Bundle Adjustment (BA) - покращує оцінку положення камери та координати 3D-точок;
– Виявлення замикання циклів (Loop Closure Detection) – якщо камера повертається у вже відвідане місце, алгоритм повинен це визначити;
– Оптимізація графа поз (Pose Graph Optimization) – якщо знайдено замикання циклу, виконується оптимізація всієї карти, щоб уникнути накопичення помилок та зробити карту узгодженою на глобальному рівні.

В статті буде розглянуто етап Bundle Adjustment (BA), оскільки він є ключовим для точного відновлення тривимірної структури сцени та параметрів камери. Від якості його виконання залежать усі наступні етапи комп'ютерного зору, такі як побудова карти глибини, трекінг та реконструкція.

У процесі побудови 3D-карти ми маємо набір камер і набір 3D-точок, які спроектовані на зображення у вигляді 2D-координат. Bundle Adjustment (BA) – це метод оптимізації, який дозволяє покращити оцінку положення камер та 3D-точок сцени, зменшуючи помилку повторної проєкції (Reprojection Error).

Математично функція помилки повторної проєкції виглядає так [4]:

$$E = \sum_{c \in C} \sum_{p,p' \in P} d(p', Q(p,c)) \tag{4}$$

де

- c – одна з камер у множині камер C;
- p – 3D-точка у множині всіх 3D-точок P;
- p' – відповідна 2D-точка, яку камера c спостерігає;
- $Q(p,c)$ – функція проєкції 3D-точки p на площину камери c;
- $d(p', Q(p,c))$ – відстань між спостереженою 2D-точкою p' і її проєкцією $Q(p,c)$.

Метою оптимізації є мінімізувати помилку між спостереженими 2D ключовими точками і проекціями 3D точок на зображення.

Сьогодні уже існують готові рішення, реалізовані у вигляді бібліотек, що спрощують використання Bundle Adjustment без необхідності розробки власної реалізації «з нуля». Метою статті є порівняння таких бібліотек, як Ceres Solver [5], GTSAM("Georgia Tech Smoothing and Mapping") [6], і SymForce [7], з точки зору їх швидкодії, точності та стійкості до шумів у даних.

Порівняння бібліотек. Порівняємо оптимізацію поз та траєкторій для бібліотек Ceres, Symforce і GTSAM на датасеті BAL (Bundle Adjustment in the Large). Джерела даних і опис датасету доступні за посиланням: [8]. Загалом структура датасету включає:

- Кількість камер – визначає кількість позицій камер у сцені.
- Кількість 3D-точок – набір тривимірних точок, які проєктуються на зображення.
- Спостереження – 2D-координати точок у зображеннях, отримані камерами.
- Параметри камер – включають внутрішні (фокусна відстань, дисторсія) та зовнішні (позиція, орієнтація) параметри.
- Формат файлів – збереження у вигляді текстових файлів із чітко структурованими списками камер, точок та спостережень.

Для тестування було використано три набори даних: Ladybug, Trafalgar Square та Dubrovnik. Для обробки, оцінки та порівняння результатів траєкторії та побудови відповідних порівняльних графіків було використано бібліотеку Evo [9].

Блок схема програмного забезпечення для тесту оптимізації наведено на рис. 2:

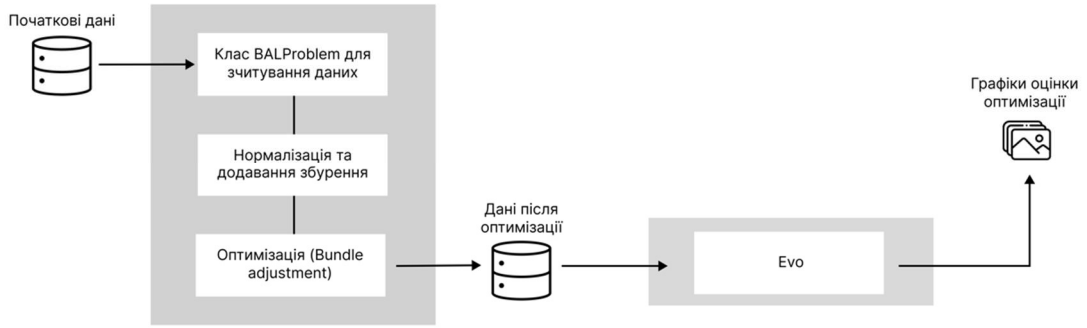

Рис. 2 – Схема роботи додатку для порівняння бібліотек

Перед оптимізацією реалізовано нормалізацію та внесення випадкових збурень значень поз та точок датасету.

Метод нормалізації масштабує геометрію так, щоб медіана абсолютного відхилення точок була рівною 100. Це забезпечує стабільність алгоритмів оптимізації. Нормалізація включає:

– Обчислення медіани координат контрольних точок;
– Визначення медіанного абсолютного відхилення;
– Масштабування всіх точок відносно медіани з коефіцієнтом, оберненим до медіанного абсолютного відхилення;
– Масштабування центрів камер за тим самим принципом.

Метод збурення вносить випадкові відхилення до даних для імітації шумів, що часто трапляються в реальних умовах. Його параметри:

– rotation_sigma – величина випадкового збурення до обертання камери;
– translation_sigma – величина збурення для трансляційної складової;
– point_sigma – випадкове збурення до координат точок.

Процес збурення включає:

– Додавання випадкових змін до координат точок, якщо point_sigma > 0;
– Додавання випадкових змін до параметрів обертання камер у формі кутів-осьових представлень (angle-axis);
– Додавання випадкових змін до трансляційної частини камер.

В даному експерименті було використано наступні значення для збурень: rotation_sigma – 0.1, translation_sigma – 0.5, point_sigma – 0.5.

Застосування цих методів гарантує, що алгоритми оцінювання параметрів працюють на коректно масштабованих і імітовано зашумлених даних, що є важливим для об’єктивного порівняння продуктивності оптимізаційних бібліотек.

Код проекту для реалізації порівнянь розміщено на GitHub: https://github.com/phd-projects-cv/opt_lib_play.

Нижче подано фрагменти коду по підготовці вхідних даних та запуск оптимізаторів Ceres та SymForce. Запуск оптимізатора бібліотеки GTSAM аналогічний до попередніх.

Фрагмент коду в якому відбувається читання даних датасету, їх нормалізація, додавання шумів та запуск bundle adjustment:

```
int main(int argc, char **argv) {
  std::string bal_file = "/Users/bohdan/CLionProjects/study/data/ladybug/problem-49-7776-pre.txt";

  google::InitGoogleLogging(argv[0]);

  BALProblem bal_problem(bal_file);
  bal_problem.Normalize();
  bal_problem.Perturb(0.1, 0.5, 0.5);
  SolveBA(bal_problem);

  return 0;
}
```

Фрагмент коду з запуском оптимізатора Ceres:

```
ceres::Solver::Options options;
options.linear_solver_type = ceres::LinearSolverType::SPARSE_SCHUR;
options.minimizer_progress_to_stdout = true;
options.max_num_iterations = 200;
ceres::Solver::Summary summary;
ceres::Solve(options, &problem, &summary);
std::cout << summary.FullReport() << "\n";
```

Фрагмент коду з запуском оптимізатора SymForce:

```
    auto params = sym::DefaultOptimizerParams();
params.verbose = true;
params.lambda_update_type = sym::lambda_update_type_t::DYNAMIC;
sym::Optimizerd optimizer{params, std::move(factors)};
const auto stats = optimizer.Optimize(values, 200);

spdlog::info("Finished in {} iterations", stats.iterations.size());
```

Всього було проведено 9 експериментів: кожен з наборів даних (Ladybug, Trafalgar Square та Dubrovnik) оброблявся бібліотеками Ceres, SymForce, GTSAM відповідно.

Кількісними критеріями роботи оптимізаційних бібліотек обрано: швидкодія, кількість ітерацій, абсолютна помилка позиції (APE) та значення помилки (Cost Initial, Cost Final). Результати оптимізації наведені в таблиці 1. Всі тести проводились на комп'ютері з Apple M3 8 core CPU.

Нижче наведена графічна інтерпретація результатів оптимізації для датасету Ladybug з використанням бібліотеки SymForce.

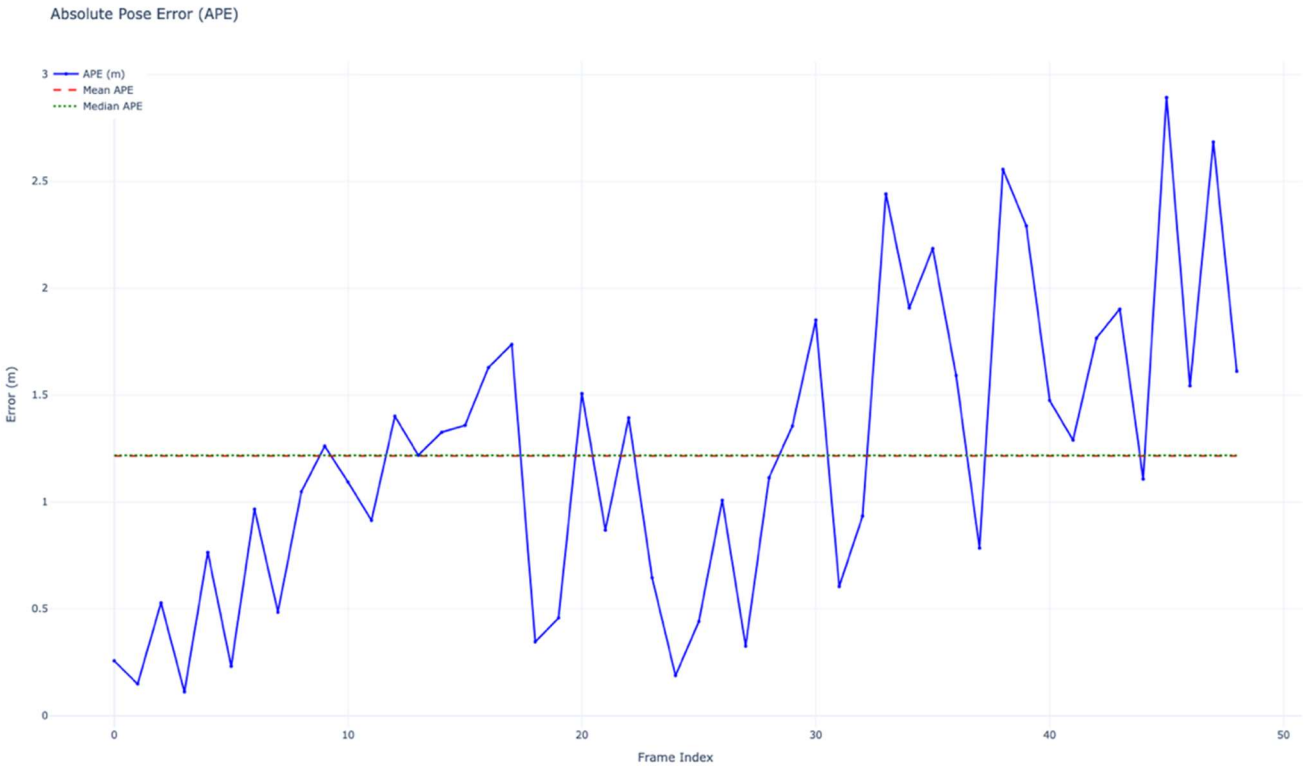

Рис. 3 – Абсолютна помилка позиції (APE) для кожного кадру:
синя крива показує абсолютну помилку позиції у метрах на кожному кадрі;
червона пунктирна лінія – середнє значення APE по всій траєкторії;
зелена точкова лінія – медіанне значення APE, що є менш чутливим до викидів.

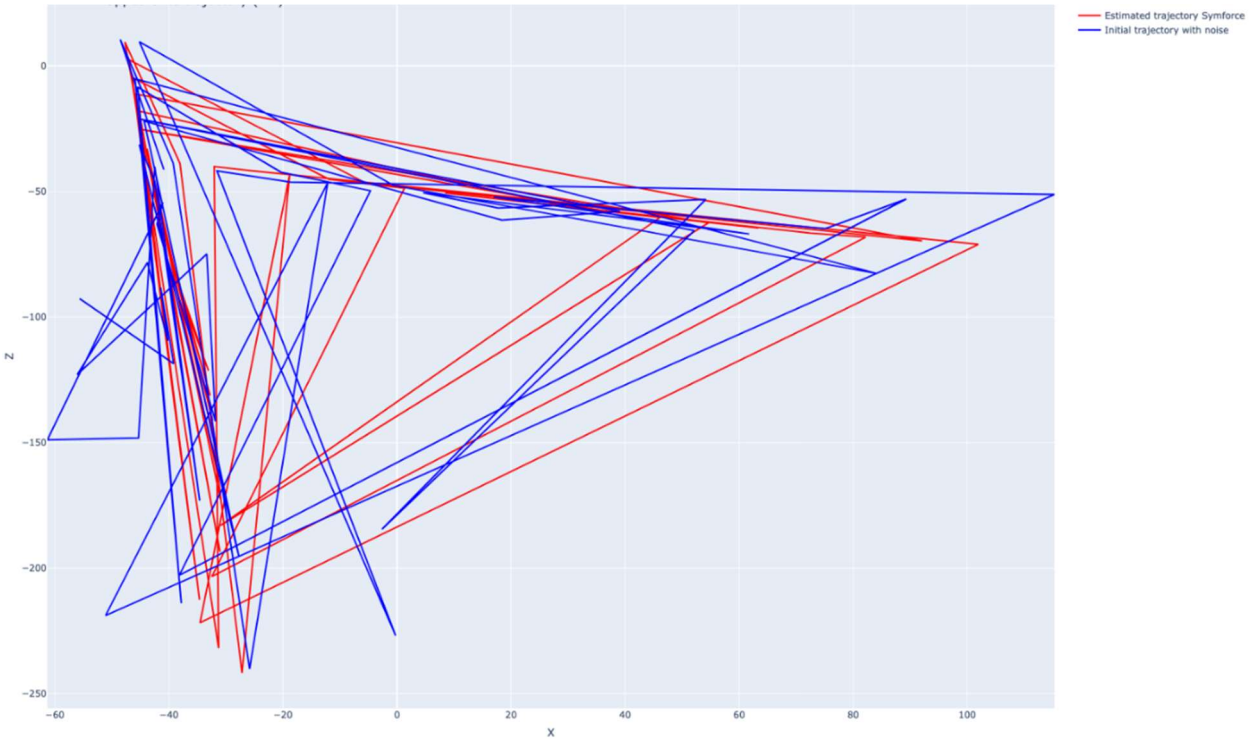

Рис. 4 – Порівняння оптимізованої та початкової траєкторії у площині X-Z:
синя лінія – початкова траєкторія з шумами перед оптимізацією;
червона лінія – оптимізована траєкторія після процедури Bundle Adjustment

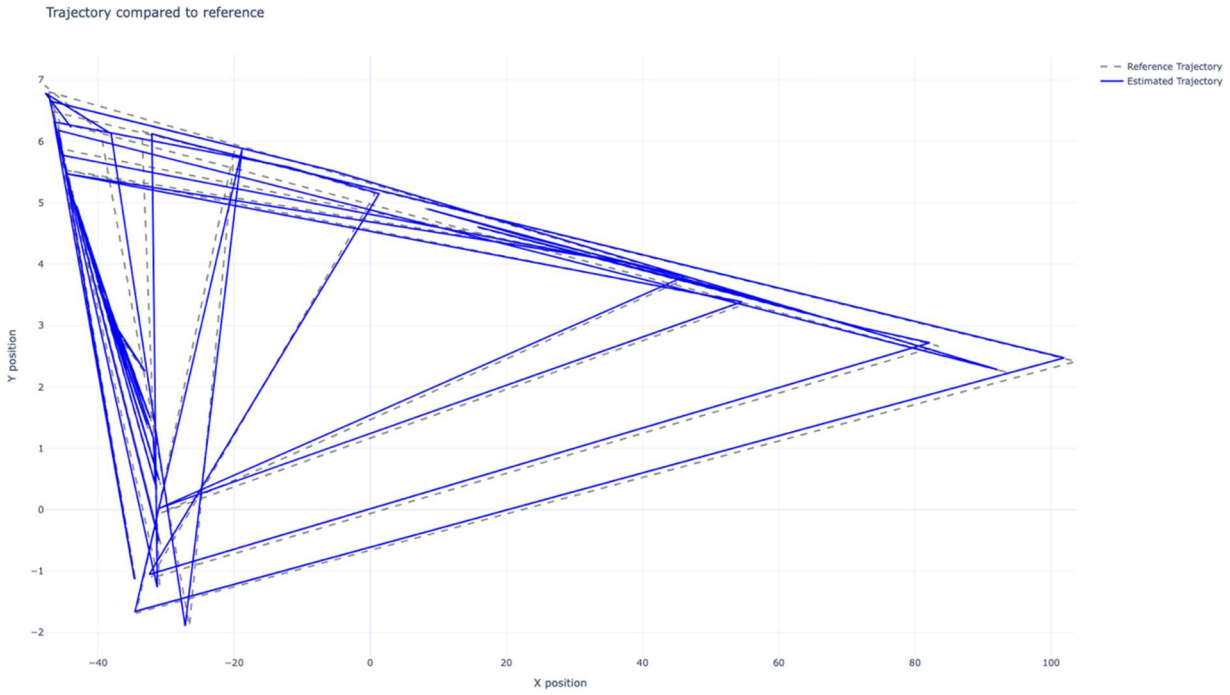

Рис. 5 – Порівняння оптимізованої та еталонної траєкторії

Таблиця 1 – Результати розв'язання задачі Bundle Adjustment

Бібліотека	Швидкодія (сек)	Кількість ітерацій	Cost Initial	Cost Final	Абсолютна помилка позиції, (м) (мін/макс)	Кількість кадрів	Кількість точок	Кількість спостережень
Ladybug								
Ceres	3.052	141	2.185e+06	8.339e+03	0.23/3.79	49	7776	31843
Symforce	2.741	105	4.339e+06	2.071e+04	0.11/2.89			
GTSAM	3.311	92	1.056e+08	1.455e+04	0.17/5.12			
Trafalgar Square								
Ceres	4.157	161	2.082e+07	1.369e+04	1.45/10.78	21	11315	36455
Symforce	2.712	95	4.118e+07	1.059e+05	0.179/5.56			
GTSAM	4.811	120	8.588e+09	3.087e+04	0.519/7.518			
Dubrovnik								
Ceres	3.046	48	2.135e+07	1.233e+04	1.276/8.03	16	22106	83718
Symforce	5.640	80	4.229e+07	3.061e+05	0.179/5.56			
GTSAM	5.833	98	4.027e+09	2.047e+04	1.026/4.599			

Висновки. В роботі розроблене програмне рішення, яке дозволяє порівнювати бібліотеки, що реалізують алгоритм Bundle Adjustment та здійснювати їх візуалізацію.

Порівняльний аналіз бібліотек Ceres, Symforce та GTSAM для трьох наборів даних (Ladybug, Trafalgar Square, Dubrovnik) дозволяє зробити наступні висновки:

1. Швидкодія

Symforce показує найкращу продуктивність у наборах Ladybug та Trafalgar Square, маючи найменший час обчислень. У наборі Dubrovnik Ceres є найшвидшою, тоді як Symforce та GTSAM значно поступаються.

2. Кількість ітерацій

Symforce потребує менше ітерацій, ніж Ceres та GTSAM, що вказує на кращу збіжність у більшості випадків. GTSAM має найменшу кількість ітерацій у наборі Ladybug, але значно програє за швидкодією.

3. Абсолютна помилка позиції (APE)

Symforce демонструє найменші помилки позиції у всіх наборах даних. Це вказує на більш точні результати оптимізації. GTSAM має найгіршу точність з найбільшими максимальними помилками.

4. Зміна значення помилки (Cost Change)

Найбільше зниження значення помилки (Cost Change) спостерігається у Symforce, що підтверджує її ефективність в оптимізації.

Спираючись на експерименти з датасетом BAL Symforce є найбільш вдалим рішенням, забезпечуючи баланс між швидкістю, точністю та кількістю ітерацій. Ceres є хорошим компромісним варіантом, особливо для Dubrovnik, де він випереджає Symforce за швидкодією. GTSAM поступається конкурентам, показуючи найбільшу абсолютну помилку та значний час виконання.

Загалом, Symforce є найкращим вибором для задач оптимізації, коли важлива як швидкодія, так і точність результатів.

Список використаної літератури

1. F. Lu and E. Milios, "Globally Consistent Range Scan Alignment for Environment Mapping," Autonomous Robots, vol. 4, no. 4, pp. 333–349, 1997.
2. Xiang Gao and Tao Zhang, "Introduction to Visual SLAM From Theory to Practice", 2021. https://github.com/gaoxiang12/slambook2
3. F. Dellaert and M. Kaess, "Square root SAM: Simultaneous localization and mapping via square root information smoothing," International Journal of Robotics Research, vol. 25, no. 12, pp. 1181–1203, 2006.
4. Visual SLAM for Robotics Perception: Building Monocular SLAM from Scratch in Python OpenCV, https://learnopencv.com/monocular-slam-in-python/
5. S. Agarwal, K. Mierle, and Others, "Ceres solver," http://ceres-solver.org.

6. F. Dellaert, V. Agrawal, A. Jain, M. Sklar, and M. Xie, "GTSAM," https://gtsam.org
7. Skydio, https://symforce.org/
8. Agarwal, S., Mierle, K., & Others. (2010). *Bundle adjustment in the large*. In *Proceedings of the European Conference on Computer Vision (ECCV)*, http://grail.cs.washington.edu/projects/bal
9. Grupp, Michael, evo: Python package for the evaluation of odometry and SLAM, https://michaelgrupp.github.io/evo/

Bohdan Bihanskyi, Dmytro Kovaliuk

COMPARISON OF SOFTWARE IMPLEMENTATIONS OF THE BUNDLE ADJUSTMENT ALGORITHM FOR THE SLAM PROBLEM

This paper investigates the problem of Simultaneous Localization and Mapping (SLAM) with a focus on the optimization stage of Bundle Adjustment (BA). This stage is crucial for accurately reconstructing the 3D structure of a scene and estimating camera parameters, which directly impact subsequent computer vision tasks such as tracking, depth map generation, and reconstruction.
Modern software libraries implementing the BA algorithm are reviewed, including Ceres Solver, GTSAM, and SymForce. An experimental comparison of these libraries is conducted based on performance criteria such as execution speed, number of iterations, Absolute Pose Error (APE), and Cost Change.
The BAL (Bundle Adjustment in the Large) dataset and three specific datasets—Ladybug, Trafalgar Square, and Dubrovnik—were used for testing. A software solution was developed to evaluate the performance of these libraries, incorporating steps such as normalization, adding random noise, and executing optimization algorithms. The visualization of solutions was performed using the Evo library.
Experimental results show that SymForce provides the best balance between speed, accuracy, and convergence. It achieved the shortest execution time for the Ladybug and Trafalgar Square datasets and exhibited the lowest Absolute Pose Error across all cases. Ceres Solver proved competitive, particularly for the Dubrovnik dataset, where it outperformed SymForce in terms of speed. In contrast, GTSAM demonstrated the worst performance across all criteria, exhibiting the highest absolute errors and the longest computation times.
Based on these findings, SymForce is the most suitable tool for SLAM and nonlinear optimization tasks where both speed and accuracy are critical. Ceres Solver can be effective in specific scenarios requiring maximum computational speed. Meanwhile, GTSAM lags behind its competitors, limiting its applicability in high-performance systems.
These conclusions will aid in selecting the appropriate optimization libraries for practical applications in robotics, autonomous vehicles, and 3D mapping.

Keywords: *SLAM, Bundle Adjustment, optimization, software, computer vision, robotics*

References

1. Lu, F., & Milios, E. (1997). Globally consistent range scan alignment for environment mapping. *Autonomous Robots, 4(4),* 333–349.
2. Gao, X., & Zhang, T. (2021). *Introduction to Visual SLAM: From Theory to Practice.* Retrieved from https://github.com/gaoxiang12/slambook2
3. Dellaert, F., & Kaess, M. (2006). Square root SAM: Simultaneous localization and mapping via square root information smoothing. *International Journal of Robotics Research, 25(12),* 1181–1203.
4. Sett, S. (2024). Visual *SLAM for Robotics Perception: Building Monocular SLAM from Scratch in Python OpenCV.* Retrieved from https://learnopencv.com/monocular-slam-in-python/
5. Agarwal, S., Mierle, K., & The Ceres Solver Team. (2023, October). Ceres Solver (Version 2.2) Software. Retrieved from https://github.com/ceres-solver/ceres-solver
6. Dellaert, F., & GTSAM Contributors. (2022, May). borglab/gtsam (Version 4.2a8) Software. Georgia Tech Borg Lab. Retrieved from https://doi.org/10.5281/zenodo.5794541.
7. Martiros, H., Miller, A., Bucki, N., Solliday, B., Kennedy, R., Zhu, J., Dang, T., Pattison, D., Zheng, H., Tomic, T., Henry, P., Cross, G., VanderMey, J., Sun, A., Wang, S., & Holtz, K. (2022). SymForce: Symbolic computation and code generation for robotics. *Proceedings of Robotics: Science and Systems.* Retrieved from https://doi.org/10.15607/RSS.2022.XVIII.04.
8. Agarwal, S., Mierle, K., & Others. (2010). Bundle adjustment in the large. In *Proceedings of the European Conference on Computer Vision (ECCV).* Retrieved from http://grail.cs.washington.edu/projects/bal
9. Grupp, M. (2017). evo: Python package for the evaluation of odometry and SLAM Software. Retrieved from https://github.com/MichaelGrupp/evo