Journal of Korea Robotics Society (2022) 17(3):373-380
https://doi.org/10.7746/jkros.2022.17.3.373 ISSN: 1975-6291 / eISSN: 2287-3961

ICP 계산속도 향상을 위한 빠른 Correspondence 매칭 방법

A Fast Correspondence Matching for Iterative Closest Point Algorithm

신 건 희[1*] · 최 재 희[2*] · 김 광 기[†]
Gunhee Shin[1*], Jaehee Choi[2*], Kwangki Kim[†]

Abstract: This paper considers a method of fast correspondence matching for iterative closest point (ICP) algorithm. In robotics, the ICP algorithm and its variants have been widely used for pose estimation by finding the translation and rotation that best align two point clouds. In computational perspectives, the main difficulty is to find the correspondence point on the reference point cloud to each observed point. Jump-table-based correspondence matching is one of the methods for reducing computation time. This paper proposes a method that corrects errors in an existing jump-table-based correspondence matching algorithm. The criterion activating the use of jump-table is modified so that the correspondence matching can be applied to the situations, such as point-cloud registration problems with highly curved surfaces, for which the existing correspondence-matching method is non-applicable. For demonstration, both hardware and simulation experiments are performed. In a hardware experiment using Hokuyo-10LX LiDAR sensor, our new algorithm shows 100% correspondence matching accuracy and 88% decrease in computation time. Using the F1TENTH simulator, the proposed algorithm is tested for an autonomous driving scenario with 2D range-bearing point cloud data and also shows 100% correspondence matching accuracy.

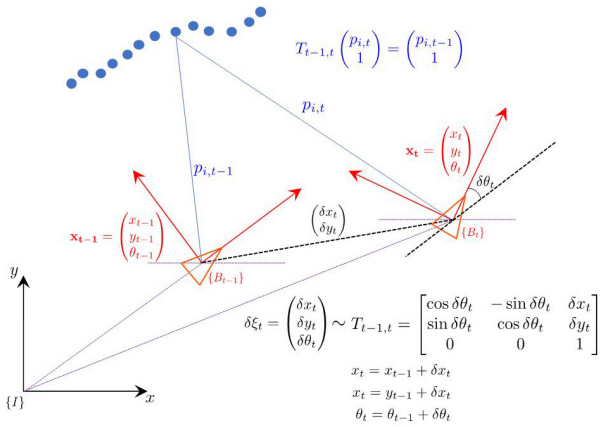
Keywords: Scan Matching, Point-Cloud Registration, Iterative Closest Point Methods, Jump Table, LiDAR, Correspondence Matching, Pose Estimation

1. 서 론

자율주행 분야 연구에는 정확하고 실시간에 가까운 측위가 보장되어야 한다. 로봇에 부착된 다양한 센서를 통해서 측위를 수행하는데 mm 수준 오차의 연속적인 거리 데이터를 얻는 라이다를 활용한 측위 기술에는 Iterative Closet Point (ICP)를 이용한 Scan Matching 방법이 많이 사용되어 왔다. 이는 Simultaneous Localization and Mapping (SLAM)의 성능향상[1] 뿐만 아니라 실내외 지도 작성 및 주변 환경 탐지와 추적에도 활용되고 있다[2,3].

Besl과 Mckay가 처음 제안한 ICP방법은 서로 다른 위치(position)와 방향(orientation)에서 측정된 두 개의 포인트 클라우드 집합들을 정합하여 서로 다른 두 시점에서의 상대적인 자세(pose)를 추정하는 기술로서, 이동로봇의 자세 추정에 적용될 수 있다[4][Fig. 1]. 이동로봇이 한 시점의 자세에서 측정하여 얻은 측정 물체 또는 지형의 포인트 클라우드를 '기준 포인트 클라우드(Reference Point Cloud, R_PC)'라고 하고, 다음 시점의 다른 자세에서 얻은 포인트 클라우드를 '변환 포인트 클라우드(Transformed Point Cloud, T_PC)'라고 할 때, 이동로봇의 자세추정 문제는 변환 포인트 클라우드를 기준 포인트 클라우드에 정합하는 병진이동과 회전이동을 구하는 문제로 정의될 수 있다. 그리고 해당되는 포인트 클라우드 정합 문제는

Received : Mar. 24. 2022; Revised : Apr. 29. 2022; Accepted : May. 7. 2022
※ This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2020R1F1A1076404)
* The authors (G. Shin and J. Choi) have equally contributed to this work
1. Undergraduate Student, Department of Electrical Engineering, Inha University, Incheon, 22212, Korea (gunmaplehee@gmail.com)
2. Undergraduate Student, Department of Mechanical Engineering, Inha University, Incheon, 22212, Korea (jahii@naver.com)
† Assistant Professor, Corresponding author: Department of Electrical Engineering, Inha University, Incheon, 22212, Korea (kwangki.kim@inha.ac.kr)



[Fig. 1] Relative pose estimation using point cloud registration and tracking in 2D space

ICP 또는 그와 유사한 변형 알고리즘들을 사용하여 해를 구할 있다. ICP 과정 중 중요한 부분은 변환 포인트 클라우드의 각 점에 대한 기준 포인트 클라우드 위의 유의미한 부분을 찾는 Correspondence 매칭 단계와 이 결과로부터 얻은 오류 함수를 최소화하는 변환 행렬을 구하는 최적화 단계이다. 두 단계가 반복적으로 수행되며 최적화의 결정변수인 변환 행렬의 변화가 임계 값보다 작으면 ICP를 종료하여 최종의 변환 행렬에서 이동로봇의 자세 변화를 표현하는 병진이동과 회전이동을 정의할 수 있다.

이동로봇의 자세 결정을 위한 측위 기술에서 중요한 성능 지표 중에 하나는 연산의 실시간성을 확보하는 것이다. [4]에서 Correspondence 단계의 과정이 ICP 알고리즘의 총 연산 시간에서 약 95%를 차지한다는 조사결과를 발표하였는데, 이는 ICP를 활용한 이동로봇 실시간 자세 결정 또는 추정을 위해서는 Correspondence 단계의 시간을 단축하는 것이 중요하다는 것을 의미한다. Correspondence 매칭 시간을 줄이기 위한 다양한 연구들이 진행되었는데, [5]에서는 변환 포인트 클라우드의 각 점에서 거리를 비교할 기준 포인트 클라우드 점들 중 후보를 선정하는 Jump Table을 이용하였고, [6,7]에서는 kd-trees, [8]에서는 다중 Z-buffer, [9]에서는 계층적 모델을 활용하여 Correspondence 단계의 시간을 단축했다.

Correspondence 매칭 시간 단축을 통한 연산 시간 감소와 함께, 또 다른 중요한 문제는 측정 데이터의 잡음과 Correspondence 매칭 오류에 취약한 최소자승법 기반의 ICP 알고리즘의 단점을 극복하는 것이다. Correspondence 단계의 결과를 바탕으로 최소자승법에 기반한 최적의 변환 행렬 계산은 Correspondence 매칭 오류에 해당하는 이상치(outlier)에 취약하다. 그러므로 위의 연구에서 속도 증대와 함께 Correspondence 매칭 오류를 줄이기 위한 매칭 알고리즘의 정확성에 대한 검증이 동반되어야 한다. 여기서 이상치에 강건한 최적화를 위한 이상치 제거 방법(e.g., RANSAC) 및 강건 추정법(e.g., M-estimation)에 대한 연구가 활발하게 진행되고 있기는 하지만, Correspondence 매칭 오류에 의한 이상치 자체를 감소시키는 것도 좋은 방법이 될 수 있다.

본 연구에서는 [5]에서 제시된 희소 탐색(sparse search)을 통하여 Correspondence 단계에서 연산시간을 단축시키는 Jump-Table 방법에 대한 정확성 측면에서의 두 가지 오류를 소개하고 이를 보완한 새로운 알고리즘을 제시한다. 제시한 알고리즘의 성능은 Hokuyo-10LX라이다를 이용한 실제 하드웨어 실험과 F1TENTH Simulator[10]를 사용한 시뮬레이션을 통해서 검증하였다.

2. 관련 연구

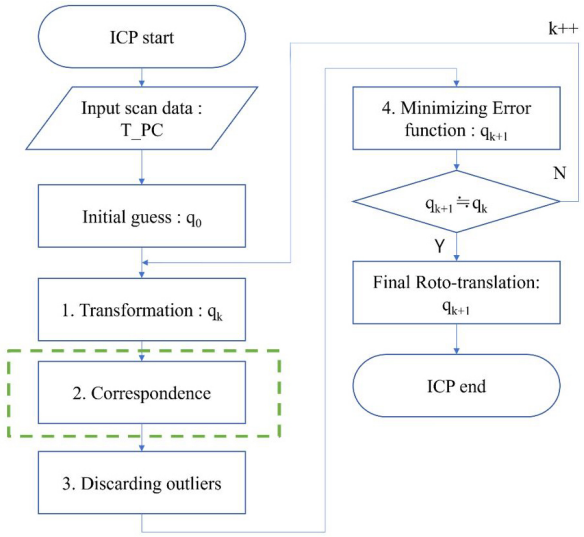
2.1 ICP (Iterative Closest Point) 알고리즘

ICP (Iterative Closest Point)는 서로 다른 위치에서 얻은 두 포인트 클라우드가 있을 때 아래 4단계를 반복하여 두 포인트 클라우드를 정합하고 마지막 단계가 끝날 때마다 새롭게 위치 차이를 추정한다. [Fig. 1]에서 볼 수 있듯이, 상대적 포즈 차이는 두 포인트 클라우드의 기준 좌표계 사이의 강체 변환 행렬(Rigid Body Transformation Matrix)로 나타낸다. 기준 포인트 클라우드의 기준 좌표계가 (X_1, Y_1), 변환 포인트 클라우드의 기준 좌표계가 (X_2, Y_2)일 때 두 좌표계 사이의 변환 행렬들의 관계는 식 (1)과 같다.

$$\begin{bmatrix}X_2\\Y_2\\1\end{bmatrix}=\mathrm{q}(\mathrm{R},\mathrm{t})\begin{bmatrix}X_1\\Y_1\\1\end{bmatrix}=\begin{bmatrix}R & t\\0 & 1\end{bmatrix}\begin{bmatrix}X_1\\Y_1\\1\end{bmatrix}=\begin{bmatrix}r_{xx} & r_{xy} & t_x\\r_{yx} & r_{yy} & t_y\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}X_1\\Y_1\\1\end{bmatrix} \quad (1)$$

포인트 클라우드는 라이다의 기준 좌표계에서 Z=0 평면에 표현되고 Z축으로 라이다의 물리적 이동이 없다고 가정한다. 4단계를 모두 수행하면 ICP를 한번 수행했다고 하며 ICP가 k번 수행되었을 때 추정한 위치 차이를 $q_k(R_k, t_k)$으로 나타낸다. [Fig. 2]의 순서도에 해당하는 ICP 알고리즘은 다음 네가지 단계를 반복한다.

1. 변환(Transformation): 이전 변환 행렬 $q_{k-1}(R_{k-1}, t_{k-1})$으로 변환 포인트 클라우드를 변환하여 새로운 포인트 클라우드를 만들고 이를 k번째 '중간 포인트 클라우드(Middle Point Cloud, M_PC)'라 한다. 변환 행렬의 초기값(q_0)은 ICP 성능에 큰 영향을 주는데 이는 Wheel Odometry 또는 관성 측정 장치(IMU)와 같은 다른 센서를 통해 예측한다. 포인트 클라우드는 라이다의 기준 좌표계에서 Z=0 평면에 나타나므로 포인트 클라우드의 점들에 일정한 규칙으로 순서를 매겼을 때 중간 포



[Fig. 2] A flow chart of ICP algorithm

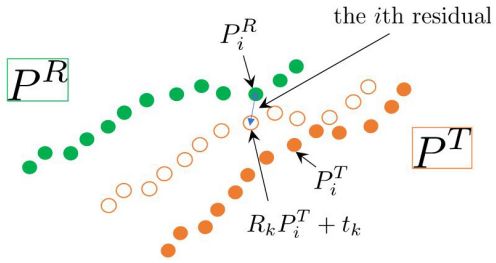
인트 클라우드의 i번째 점은 $P_i^M = \begin{bmatrix} x_i^M \\ y_i^M \end{bmatrix}$ 으로 표현된다. 그리고 q_k를 이용하여 변환 포인트 클라우드를 변환하여 중간 포인트 클라우드를 구하는 식은 식 (2)와 같다.

$$\Sigma_i P_i^M = \Sigma_i \begin{bmatrix} x_i^M \\ y_i^M \\ 1 \end{bmatrix} \triangleq \Sigma_i \left\{ q_{k-1}(R_{k-1}, t_{k-1}) \begin{bmatrix} x_i^T \\ y_i^T \\ 1 \end{bmatrix} \right\} \quad (2)$$
$$= \Sigma_i (R_{k-1} P_i^T + t_{k-1})$$

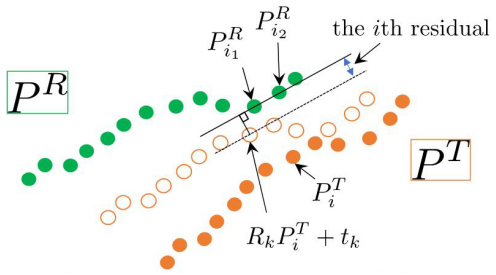
2. 대응 관계(Correspondence): k번째 중간 포인트 클라우드의 각 점에 대응하는 기준 포인트 클라우드 위의 유의미한 부분을 찾는다. 여기서 유의미한 부분(Segment)은 유클리드 공간에서 거리가 가장 가까운 점을 기반으로 형성되는데 [2]의 점-점(point-to-point) ICP의 경우 가장 가까운 점이 되며 [5]의 점-선(point-to-line) ICP의 경우 가장 가까운 점 두개를 이은 직선이 된다. 기준 포인트 클라우드 위의 유의미한 부분은 대응하는 변환 포인트 클라우드의 점(P_i^T)과 함께 배열로 저장되며 점-점의 경우 $C_k : \langle P_i^T, P_{i_1}^R \rangle$, 점-선의 경우 $C_k : \langle P_i^T, P_{i_1}^R, P_{i_2}^R \rangle$이다. $P_{i_1}^R$는 P_i^M에서 가장 가까이에 위치한 기준 포인트 클라우드의 한 점이고 $P_{i_2}^R$는 두번째로 가까운 점이다.

3. 이상치 제거(Outlier rejection): ICP는 correspondence결과를 바탕으로 최소자승법(Least Squares Method)를 통해 해를 구한다. 최소자승법은 이상치에 취약하므로 Correspondence 결과에 가중치를 부여하는 M-estimation 방법을 사용하거나 이상치를 판별하여 제거하는 RANSAC 방법이 많이 사용된다.

4. 오차함수 최소화(Minimizing Error Function): 이상치를 제거한 Correspondence 결과를 이용하여 오차함수를 구한 뒤



(a) Point-to-point correspondence error (or residual) criterion associated with the cost function (3)



(b) Point-to-line correspondence error (or residual) criterion associated with the cost function (4)

[Fig. 3] Schematic diagrams of two different residual and cost functions: (a) Point-to-point and (b) Point-to-line

최소자승법으로 이를 최소화하는 q_k를 구한다. ICP 종류에 따라 오차함수의 형태가 구분된다. [Fig. 3(a)]의 점-점의 경우 강체 변환된 P_i^T와 $P_{i_1}^R$ 사이의 유클리디안 거리이고 [Fig. 3(b)]의 점-선의 경우 강체 변환된 P_i^T에서 $P_{i_1}^R$,$P_{i_2}^R$ 두 점을 이은 직선까지의 수직 거리이다. 현 단계에서 구한 q_k와 이전 단계에서 구한 q_{k-1}의 차이를 통해 ICP 종료 여부를 판별하는데 서로 다른 위치에서 얻은 포인트 클라우드가 완벽하게 정합하는 것은 불가능하므로 일정 임계값 ϵ를 이용하여 판별한다. q_k와 q_{k-1}의 차이가 ϵ보다 크면 1번으로 돌아가 q_k를 이용하여 다시 변환하고 차이가 작다면 ICP를 종료하고 최종 변환 행렬을 q_k로 확정한다. ϵ의 크기는 정합의 정도와 정확도 사이의 절충을 표현한다. ϵ의 크기가 크면 정합이 충분히 되지 않고 ϵ 값이 작으면 올바르지 않은 Correspondence이 발생하여 최종 정합이 어긋날 수 있다.

$$E_{p2p}(q_k, C_k) = \Sigma_i (R_k P_i^T + t_k - P_1^{i_1})^2 \quad (3)$$

$$E_{p2l}(q_k, C_k) = \Sigma_i (n_i^T \left[R_k P_i^T + t_k - P_1^{i_1} \right])^2 \quad (4)$$

ICP 알고리즘은 변환 포인트 클라우드의 각 점에 대응하는 기준 포인트 클라우드의 유의미한 부분을 찾고 그 사이의 거리 합을 최소로 만드는 변환 행렬 q(R,t)를 구하는 과정을 반복적으로 수행한다. ICP를 한번 수행했을 때 4단계를 하나의 식

으로 표현하면 식 (5)와 같다. S_i^{ref}는 P_i^T에 대한 기존 포인트 클라우드 위의 유의미한 부분이며 $\overleftarrow{(S_i^{ref},)}$는 유클리드 공간에서 S_i^{ref} 위로의 사영을 의미한다.

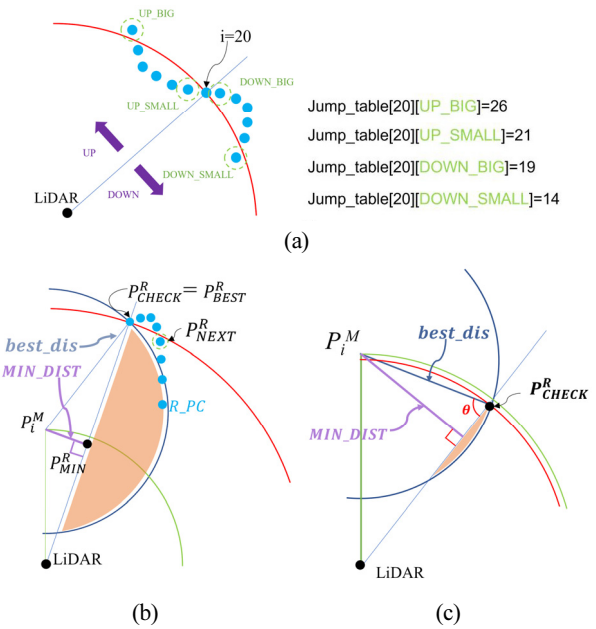
$$\min_{q_k} \sum_i \| P_i^T \times q_k - \overleftarrow{(S_i^{ref}, P_i^T \times q_{k-1})} \|^2 \quad (5)$$

2.2 Jump Table을 이용한 Correspondence

A.Censi는 Correspondence의 시간을 Jump Table을 사용하여 단축했다[5]. Correspondence는 중간 포인트 클라우드의 각 점에 대해 기준 포인트 클라우드를 탐색하며 가장 가까운 점을 찾는다. [5]에서는 기준 포인트 클라우드 위의 모든 점을 차례대로 탐색하지 않고 Jump Table을 사용하여 기존 포인트 클라우드의 점들을 점프하며 탐색한다.

Jump Table은 원점(라이다)에서 기준 포인트 클라우드의 각 점까지의 거리 비교를 통해 한 점에 대해 서로 다른 네 개의 점을 설정한다(up_small, up_big, down_small, down_big). 기준 포인트 클라우드 한 점에 대해 반 시계 방향을 위(up), 시계 방향을 아래(down)로 할 때, 최초로 거리가 큰 점을 big, 거리가 작은 점을 small로 설정한다. 예를 들어, 어떤 기준 포인트 클라우드의 20번째 점의 Jump Table은 [Fig. 4(a)]와 같이 설정된다.

중간 포인트 클라우드의 한 점(P_i^M)에서 가장 가까운 기준 포인트 클라우드의 한 점을 찾는 상황에서 현재 탐색하고 있는



[Fig. 4] (a) Example of jump table (b) Finding next check point when searching direction is down (c) Exception situation of existing Jump table-based Correspondence algorithm

점(P^R_{CHECK})이 가장 가까운 거리(best_dis)를 갖는 점(P^R_{Best})으로 업데이트 된 경우는 [Fig. 4(b)]와 같다.

아래 방향으로 탐색할 때 새롭게 best_dis가 업데이트 될 수 있는 경우는 기준 포인트 클라우드의 점이 주황색 구역에 존재할 때이다. 위 상황에서 [5]는 Jump Table을 이용하여 아래 방향에 존재하는 점들을 차례대로 탐색하지 않고 주황색 영역에 존재할 가능성이 있는 점들만 Jump하며 탐색한다. 그리고 P_i^M에서 원점과 기준 포인트 클라우드의 한 점을 이은 직선에 수직으로 내린 점(P^R_{MIN})까지의 거리(MIN_DIST)가 현재의 best_dis보다 크게 되면 탐색을 중지하고 처음 탐색을 시작한 점(P^R_{START})로 돌아가 반대 방향으로 탐색을 시작한다. 여기서 P^R_{START}은 원점에서 P_i^M으로의 방향 벡터 위에 존재하는 기준 포인트 클라우드 위의 한 점이다. [5]에서 ICP를 수행하기 전에 기준 포인트 클라우드의 각 점에 대한 Jump Table을 저장하는 시간은 전체 ICP 시간의 1%를 차지하며 Jump Table을 사용했을 때 기준 포인트 클라우드 위의 모든 점을 차례대로 탐색하는 일반적인 방법보다 시간을 82% 단축한 결과를 보였다.

3. 새로운 Jump Table 기반 Correspondence

3.1 Next check point 결정

[Fig. 4(b)]의 P^R_{CHECK}에서 아래 방향으로 탐색할 때 [5]는 다음으로 탐색할 점(P^R_{NEXT})을 P^R_{CHECK}의 Jump Table에서 선택하는데 그 기준은 원점에서 P_i^M의 거리($P_i^M.r$)와 P^R_{CHECK}의 거리($P^R_{CHECK}.r$)의 대소이다[Table 1(a)]. [Fig. 4(b)]에서는 아래 방향으로 탐색하고 있으며 $P_i^M.r$이 $P^R_{CHECK}.r$보다 작으므로 P^R_{NEXT}는 P^R_{CHECK}의 down _small이 되어 주황색 영역에 존재할 수 있는 점을 확인하게 되는 것이다.

[Table 1] (a) Existing rule presented in [5] to select P^R_{NEXT} (b) Our proposed rule to select P^R_{NEXT}

	(a)		(b)	
	Status	P^R_{NEXT}	Status	P^R_{NEXT}
UP	$P_i^M.r > P^R_{CHECK}.r$	up_small	$\theta < \frac{\pi}{2}$	up_small
UP	$P_i^M.r < P^R_{CHECK}.r$	up_big	$\theta > \frac{\pi}{2}$	up_big
DOWN	$P_i^M.r > P^R_{CHECK}.r$	down_small	$\theta < \frac{\pi}{2}$	down_small
DOWN	$P_i^M.r < P^R_{CHECK}.r$	down_big	$\theta > \frac{\pi}{2}$	down_big

대부분의 상황에서는 [Table 1(a)]의 기준이 합리적이지만 P^R_{CHECK}이 $P^R_{MIN}.r<P^R_{CHECK}.r<P^M_i.r$인 구간에 존재할 경우 [Fig. 4(c)]는 합리적이지 않다. $P^R_{MIN}.r$은 원점에서부터 P^R_{MIN}까지의 거리이다. best_dis가 업데이트 되기 위해서는 P^R_{NEXT}이 주황색 영역에 존재해야 하며, [Fig. 4(c)]를 보았을 때, P^R_{CHECK}의 Jump Table에서 down_small이 선택되어야 하지만 [5]의 기준으로는 $P^M_i.r$가 $P^R_{CHECK}.r$ 보다 크므로 down_big을 선택한다.

[5]에서 제안된 Jump Table에서 P^R_{NEXT}을 선택하는 기준에는 오류가 있으며 우리는 P^R_{CHECK}에서 바라보는 원점과 P^w_i의 사이각(θ)을 이용하여 새로운 기준을 [Table 1(b)]와 같이 제시한다. θ이 예각일 경우 best_dis가 업데이트 될 수 있는 영역이 P^R_{CHECK}의 원점 방향에 존재하므로 P^R_{NEXT}의 small을 선택하고 둔각일 경우 best_dis가 업데이트 될 수 있는 영역이 P^R_{CHECK}의 바깥 방향에 존재하므로 big을 선택한다.

3.2 360°에 유효한 Jump Table

[5]의 경우, P^R_{NEXT}가 아래 방향 탐색 시 기준 포인트 클라우드의 시작점(0^{th}), 위 방향 탐색 시 기준 포인트 클라우드의 끝점(n^{th})에 도달할 경우 해당 방향으로의 탐색을 중지하므로, 라이다의 시야각이 360° 에 근접할 때 오류가 발생한다. P^R_{START}가 0^{th}에 가까울 경우 best_dis가 0^{th}을 넘어 n^{th} 부근에서 업데이트 될 수 있기 때문이다. 우리는 P^R_{NEXT}가 0^{th}에 도달하면 n^{th}으로, n^{th}에 도달하면 0^{th} 넘겨주고, P^R_{CHECK}가 P^R_{START}에서 180° 떨어진 점에 도달했을 때 탐색을 중지하도록 했다. 이를 통해 시야각에 제한을 받지 않고 모든 범위에서 Jump Table을 사용할 수 있다.

3.3 Pseudo Code 설명

우리가 새로 고안한 알고리즘의 Pseudo Code는 [Algorithm 1]과 같으며 사용된 기호의 의미는 [Table 2]와 같다.

[Algorithm 1]의 주요 구문의 의미는 다음과 같다:

line 1 : 포인트 클라우드 P^M 위의 모든 점에 대해 탐색.

line 9 : up_stopped 과 down_stopped 모두 참이면 탐색 종료.

line 10 : 탐색이 중지된 방향의 반대쪽으로 탐색하도록 설정.

line 12~14 : P^R_{CHECK}이 끝지점에 도달하면 반대 끝지점으로 넘어가 같은 방향으로 탐색 지속.

line 24 : best_dis가 min_dist보다 크면 더 이상 해당 방향으로 탐색하는 것이 무의미하므로 탐색 종료.

line 26~27 : $\theta_{criteria}$를 코사인 제 2 법칙을 이용하여 구함.

line 30~34 : Table 1(1)를 기준으로 $\theta_{criteria}$의 값에 따라 다음으로 탐색할 점을 결정.

[Algorithm 1] Pseudo Code of our jump table algorithm

```
1   for(each point P_i^M in P^M){
2     int P_BEST^R; double best_dis;
3     int start_index = int(P_i^M.θ/angle_increment);
4     int P_DOWN CHECK^R=start_index;
5     int P_UP CHECK^R=start_index+1;
6     double dist_up; //shortest_distance_in_up
7     double dist_down; //shortest_distance_in_down
8     bool up_stopped=False; bool down_stopped =False;
9       while(!(up_stopped && down_stopped)){
10        bool now_up = up_stopped ? 0 : down_stopped ? 1 : 0
11      if(!now_up){ //checking down direction
12        if(P_DOWN CHECK^R<0){
13              P_DOWN CHECK^R=n; continue;
14        } //n is the last index of reference point cloud
15          //When P_DOWN CHECK^R is out of range in down direction,
16          //P_DOWN CHECK^R becomes 'n' to check the opposite side
17        dist_down=|P_i^M-P_DOWN CHECK^R|
18          //The distance from P_i^M to P_DOWN CHECK^R point
19        if(dist_down < best_dis){best=P_DOWN CHECK^R;
20                                 best_dis= dist_down;}
21        //update best point if the distance between checking point and
          P_i^M is shorter
22          double Δθ = P_i^M.θ - P_DOWN CHECK^R.θ
23                  double min_dist = |sin(Δθ)* P_i^M.r|
24        if(min_dist>best_dis){down_stopped=True; continue;}
25        // stop checking when min_dist is out of best distance
26        double θ_criteria=
27  cos^-1((|P_i^M- P_DOWN CHECK^R|^2 + P_DOWN CHECK^R.r^2 - P_i^M.r^2)/(2|P_i^M- P_DOWN CHECK^R| * P_DOWN CHECK^R.r))
28        // obtain θ_criteria, angle between line |P_i^M- P_DOWN CHECK^R|
29          and line P_i^M.r
30        if(θ_criteria > π/2){
31          P_DOWN CHECK^R=Jump_table[P_DOWN CHECK^R][Down_Big];
32        }else if(θ_criteria < π/2){
33          P_DOWN CHECK^R=Jump_table[P_DOWN CHECK^R][Down_Small];
34        }
35      }// if(!now_up)
36      if(now_up){ // same checking process in up direction
37          ··· }
38      }
39
40  } //for each point P_i^M
41
42
```

[Table 2] Symbols used in [Algorithm 1]

Symbol	meaning
P^M	Middle point cloud (M_PC)
P_i^M	The ith point in M_PC
P^R	Reference point cloud (R_PC)
P_{BEST}^R	The closest point in R_PC from current P_i^M
best_dis	Distance from P_i^M to P_{BEST}^R
$P_i^M.\theta$	The angle between -x axis and line from origin (LiDAR) to P_i^M
angle_increment	The unit angle of the point cloud which LiDAR reads
start_index	The point in R_PC which starts to search
dist_up/down	The distance from P_i^M to current checking point in UP/DOWN direction
up_stopped/ down_stopped	The index which determines whether to stop searching UP/DOWN direction
now_up	The index which determines whether to search UP direction
$P_{UP/DOWNCHECK}^R$	The point in R_PC which currently searching in UP/DOWN direction
$\Delta\theta$	Angle between two straight lines. First line from origin to P_i^M, second line from origin to P_{CHECK}^R
$P_i^M.r$	Distance from origin to P_i^M
min_dist	Length of the perpendicular from P_i^M to $P_{CHECK}^R.r$ (Refer to [Fig. 4(b)][Fig. 4(c)])
$\theta_{criteria}$	Angle between two straight lines. First from P_{CHECK}^R to P_i^M, second line from P_{CHECK}^R to the origin(θ from [Fig. 4(c)])

4. 실험 및 결과

4.1 실험

Hokuyo-10LX와 F1TENTH Simulator를 각각 활용하여 두 번의 실험을 진행했다[Fig. 5]. 먼저, 270°의 시야각을 가진 Hokuyo-10LX 라이다를 F1TENTH 경주차에 부착하고 인하대학교 하이테크 건물 6층에서 구동하여 실시간으로 데이터를 받았다. 이 데이터를 우리의 Jump Table 기반 Correspondence, [5]의 Jump Table 기반 Correspondence, 그리고 일반적인 Correspondence 알고리즘(Naïve algorithm)에 동일하게 입력하고 비교했다. 일반적인 Correspondence는 P_i^M에서 기준 포인트 클라우드의 모든 점들과 거리를 빠짐없이 비교하여 P_{BEST}^R 점을 찾는다. 우리의 알고리즘에서 수행된 중간 포인트 클라우드의 모든 점들에 대한 P_{BEST}^R가 일반적인 Correspondence에서 수행된 것과 일치한다면 Correspondence가 정확히 수행된 것으로 판단했다. Correspondence 이후 단계인 이상치 제거 단계는 생략했으며 오류함수 최소화 단계는 [5]의 PL-ICP의 방법을 사용했다.

Jump Table이 360°에 대해 유효한 우리의 알고리즘의 성능을 270°시야각의 Hokuyo-10LX으로 확인하기에는 한계가 있어 360°시야각의 라이다가 구현된 F1TENTH Simulator[10]에서 추가로 실험했다. [Fig. 5(b)]와 같이 Simulator에서 제공하는 주행환경에서 차량이 한 쪽 벽면을 따라가며 얻은 라이다 데이터를 이용하여 같은 비교를 진행하였다. 차량의 선속도와 각속도는 각각 1.8 m/s와 0.3 rad/s이며 차량의 가로 길이와 세로 길이는 각각 0.33 m와 0.2 m이다.

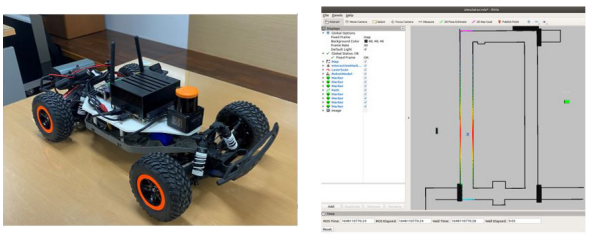
두 실험에서 사용된 소스코드는 https://github.com/jahii/F1tenth의 Experiment 브랜치 안에 scan_matching_skeleton/src/correspond.cpp에서 확인할 수 있다. 또한, 실험 영상은 https://youtu.be/VvvXB3yQ_Wg 에서 확인할 수 있다.

4.2 실험 결과

Hokuyo-10LX를 이용한 첫번째 실험에서는 대부분의 경우 중간 포인트 클라우드의 각 점에 대한 P_{BEST}^R이 세 개의 알고리즘에서 일치했지만 급격한 커브 구간에서 [Fig. 4(c)] 상황으로 인해 [5]의 알고리즘에서 일반적인 Correspondence와 일치하지 않는 점들이 존재했다. 반면에 우리의 알고리즘은 일반적인 Correspondence와 모두 일치했다. [Fig. 6]는 우리 알고리즘의 각 Correspondence 단계에서 중간 포인트 클라우드의 모든 점들이 P_{BEST}^R를 찾기 위해 탐색한 기준 포인트 클라우드의 점의 개수를 나타낸 그래프이며, [Fig. 7]는 일반적인 Correspondence와 우리 알고리즘에서 위의 탐색 과정에 소요되는 시간을 비교한 그래프이다. 기준, 중간 포인트 클라우드의 점의 개수는 각각 1080개(n=1080)이다. 각 Correspondence 과정에서 일반적인 Correspondence의 경우, 중간 포인트 클라우드의 각 점에 대해 기준 포인트 클라우드의 모든 점을 탐색하면서 P_{BEST}^R를 찾기 때문에 총 1080^2=1,166,400개의 점을 탐색한다. 그러나 우리의 알고리즘은 80% 이상의 Correspondence 과정에서 22,000개 이하로 탐색했으며 일반적인 Correspondence보다 평균적

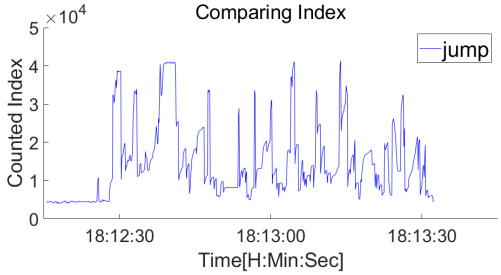
[Table 3] Mean value of number of searched points and searching time in Naïve correspondence & our jump table algorithm

	Our new jump table algorithm(①)	Naïve correspondence algorithm(②)	(1-①/②) *100%
The number of search points	14,178	1,166,400	98.784%
Computation time [ms]	70.1417	584.008	87.990%

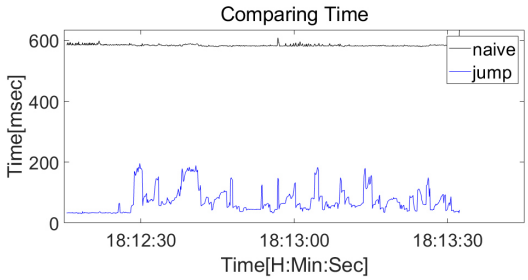


(a) (b)

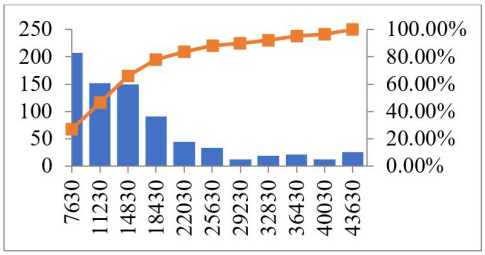
[Fig. 5] (a) F1TENTH racecar with Hokuyo-10LX; (b) F1TENTH Simulator



[Fig. 6] The number of searched points in our algorithm



[Fig. 7] Time spent processing the correspondence in Naïve algorithm and our algorithm



[Fig. 8] Histogram of the number of checking points in jump table algorithm (x-axis: [counts])

으로 95% 이상 적게 탐색했다[Fig. 8][Table 3]. 시간 측면에서 비교하면, 일반적인 Correspondence는 각 Correspondence 과정이 평균적으로 584.01 ms가 걸리고 약 90%의 과정이 580 ms~590 ms에 해당하여 일정한 시간이 소요되었다[Table 3][Fig. 9]. 반면에 우리 알고리즘은 평균적으로 70.142 ms가 걸렸으며, 약 90%의 Correspondence 과정이 130 ms이하로 소요되며 안정적인 시간 분포를 보였다[Table 3][Fig. 9].

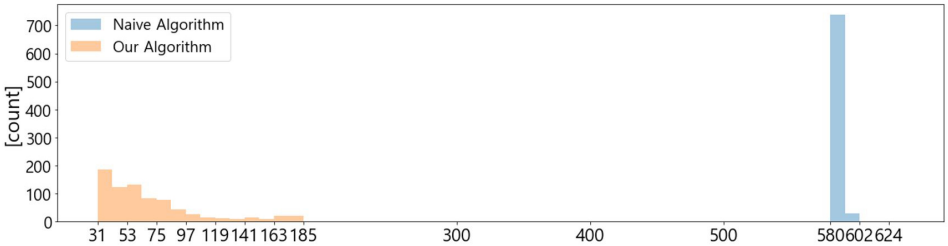
F1TENTH Simulator를 이용한 두 번째 실험에서는 P^R_{START}가 0^{th} 또는 n^{th}에 가까울 때 n^{th} 또는 0^{th}로 P^R_{NEXT}을 넘기지 못해 [5]의 알고리즘에서 일반적인 Correspondence와 일치하지 않는 점이 존재하는 반면 우리의 알고리즘에서는 모두 일치했다. 중간 포인트 클라우드의 모든 점들이 P^R_{BEST}를 찾기 위해 탐색한 기준 포인트 클라우드의 점의 개수와 탐색하는 데에 걸리는 시간을 비교했을 때 첫번째 실험과 같은 경향을 보였다.

5. 결 론

본 연구에서는 [5]의 Jump Table 기반 Correspondence에서 오류를 도출하고 이를 보완하면서 일반적인 Correspondence 방법보다 시간이 크게 단축된 새로운 Jump Table 기반 Correspondence을 제안한다. F1tenth Simulator와 Hokuyo-10LX를 활용한 실험을 통해 성능향상을 확인하였다. 향후, ICP의 Correspondence 시간을 줄이는 기존의 방법들과 본 연구논문에서 제안한 방법의 성능을 직접적으로 비교하는 연구를 진행할 예정이다.

References

[1] D. Hahnel, W. Burgard, D. Fox, and S. Thrun, “An efficient fastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements,” *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, Las Vegas, NV, USA, 2003, DOI: 10.1109/IROS.2003.1250629.



[Fig. 9] Histogram of time spent processing the correspondence in Naïve algorithm and our algorithm (x-axis: [msec])

[2] J. Minguez, L. Montesano, and L. Montano, “An architecture for sensor-based navigation in realistic dynamic and troublesome scenarios,” *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH 37566)*, Sendai, Japan, 2004, DOI: 10.1109/IROS.2004.1389825.

[3] D. Schulz, W. Burgard, D. Fox, and A. B. Cremers, “Tracking multiple moving targets with a mobile robot using particle filters and statistical data association,” *2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, Seoul, South Korea, 2001, DOI: 10.1109/ROBOT.2001.932850.

[4] P. J. Besl and Neil D. McKay, “A method for registration of 3-D shapes,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239-256, Feb., 1992, DOI: 10.1109/34.121791.

[5] A. Censi, “An ICP variant using a point-to-line metric,” *2008 IEEE International Conference on Robotics and Automation*, Pasadena, CA, USA, 2008, DOI: 10.1109/ROBOT.2008.4543181.

[6] M. Greenspan and M. Yurick, “Approximate k-d tree search for efficient ICP,” *Fourth International Conference on 3-D Digital Imaging and Modeling, 2003*, Banff, AB, Canada, 2003, DOI: 10.1109/IM.2003.1240280.

[7] W.-S. Choi, Y.-S. Kim, S.-Y. Oh, and J. Lee, “Fast Iterative Closest Point framework for 3D LIDAR data in intelligent vehicle,” *2012 IEEE Intelligent Vehicles Symposium*, Madrid, Spain, 2012, DOI: 10.1109/IVS.2012.6232293.

[8] R. Benjemaa and F. Schmitt, “Fast global registration of 3D sampled surfaces using a multi-z-buffer technique,” *Proceedings. International Conference on Recent Advances in 3-D Digital Imaging and Modeling (Cat. No.97TB100134)*, 1997, pp. 113-120, DOI: 10.1109/IM.1997.603856.

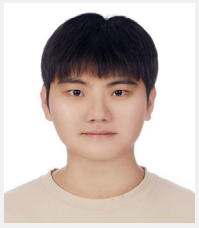
[9] D. Kim and D. Kim, “A Fast ICP Algorithm for 3-D Human Body Motion Tracking,” *IEEE Signal Processing Letters*, vol. 17, no. 4, pp. 402-405, April 2010, DOI: 10.1109/LSP.2009.2039888.

[10] F1TENTH Simulator, F1TENTH Racing Community, [Online], https://github.com/f1tenth/f1tenth_gym_ros, Accessed: May 27, 2022.

신 건 희

2017~현재 인하대학교 전기공학과(학사)

관심분야: 컴퓨터 비전, SLAM, 자율주행



최 재 희

2016~현재 인하대학교 기계공학과(학사)

관심분야: 로보틱스, 경로 추정, 자율주행

김 광 기

2013 일리노이 주립대학교(UIUC) 박사

2013~2015 조지아 공과대학(Georgia Institute of Technology) 박사후연구원

2016~2017 현대자동차 책임연구원

2017~현재 인하대학교 전기공학과 조교수

관심분야: 제어이론, 최적화이론, 최적제어, 계산과학, 로보틱스, 센서퓨전, 공간인지