Scan Match using ICP(Iterative Closest Point)

1. Abstract

Reference: A. Censi, "An ICP variant using a point-to-line metric," 2008 IEEE International Conference on Robotics and Automation, 2008, pp. 19-25, doi: 10.1109/ROBOT.2008.4543181.

위 논문은 Point to Line ICP를 제시하고 Vanilla ICP 또는 Metric Based ICP 등 다른 ICP와 비교하여 PL-ICP의 우수성을 보여준다. 위 논문에서 제공하는 오픈소스와 UPENN에서 제공하는 F1TENTH simulator 오픈소스를 참고하여 PL-ICP를 실습하였다. 그 과정에서 ICP 알고리즘이 동작하는 속도에 주목했다. ICP는 두개의 다른 PointCloud를 정합 하는 알고리즘인데 두개의 PointCloud가 짧은 시간 간격으로 생길 때 도출되는 roto-translation metric를 누적하여 얻은 pose와 ground truth pose의 오차가 적다는 것을 확인했다.

다른 ICP 관련 논문에서 ICP 과정에서 전체 running time 중 Correspondence를 찾는 과정이 95%를 차지한다고 발표했다. 그래서 PL-ICP 논문에서 제시한 Correspondence를 찾는 시간을 줄여주는 Jump_table 방법론을 공부했다. 논문과 논문에서 제공한 코드를 참고하여 F1TENTH simulator에서 Jump_table 방법론을 구현하고 확인했다. 그런데 시간 측면에서는 큰 감소를 보였지만 정확도 측면에서 문제가 발생했다. 대부분의 상황에서 Naïve Correspondence, 모든 Ref Pontcloud의 점과 비교하는 방법론, 와 index가 일치했지만 급격한 커브 구간에서는 그렇지 않는 것을 확인했다. 논문에서 제시한 방법론을 구현하는 과정에서 오류가 발생했다 생각하고 오랜 시간 확인하고 수정했지만 계속 같은 오류가 발생했고 코드에는 이상이 없다는 결론을 내렸다. 결국, 논문에서 제시한 Jump Table 알고리즘에 오류가 있다고 판단하여 다양한 상황에서 기하학적으로 접근한 결과 문에서 제시한 방법론을 사용했을 때 예외가 있다는 것을 발견했다. 그 예외를 다른 상황과 동일하게 처리할 수 있고 다른 작은 문제들까지 모두 해결한 발전된 Jump table 아이디어를 개발했다. F1TENTH simulator를 통해 simulatoin한 결과 naïve Correspondence와 index가 모든 상황에서 일치하는 것을 확인할 수 있었고 시간 측면에서도 현저한 감소를 확인할 수 있었다.

첨부한 코드에서 src폴더에 있는 scan_match.cpp, correspond.cpp는 직접 프로그래밍을 한 것이고 나머지는 UPENN F1TENTH에서 제공한 오픈소스이다.

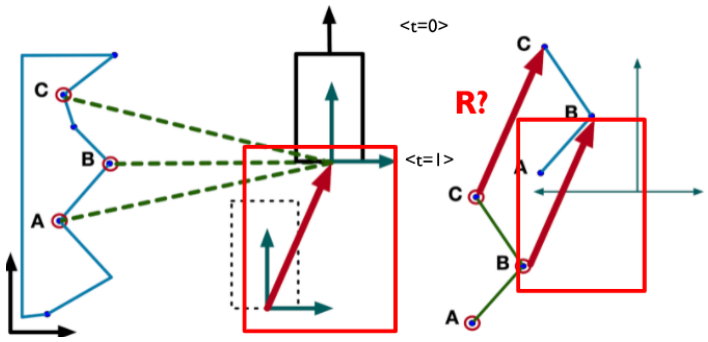
2. Concept

1. Scan Matching

- Localization의 한 가지 접근 방법으로, 로봇이 시간에 따라 이동하면서 라이다 같은 센서에서 얻은 거리 데이터의 집합을 이용하여 로봇이 이동한 Roto-Translation Matric을 얻는다.

- Scan Matching에서 Odometry 정보는 사용하지 않지만 Odometry 정보와 함께 사용되면 (ex, Extended Kalman Filter) Localization의 정확성을 향상시킬 수 있다.
- 아래 사진에서 A,B,C(landmark)의 측정값을 정확하게 알면 문제는 쉬워지지만 우리는 어떤 측정값이 landmark를 의미하는 지 알지 못한다. 그러므로 가장 가까운 점을 찾는 Correspondence match를 찾고 반복적으로 Best Transform R을 구한다.
- 반복적으로 Best Transform R을 구하는 과정은 다음과 같다

1. Make some initial "guess" of R(current guess)
2. For each point in new scan (t=k+1), find closest point in previous set(t=k), (correspondence search)
3. Make another "better guess" of R (next guess)
4. Set next guess to be the current guess, repeat steps 2-4 until converges



<FIG.1 Roto-Translation Matric>

2.ICP

- Reference Surface S^{ref}, Set of points p_i, Roto-Translation q=(t,θ)라고 할 때, q에 따라 이동한 p_i와 S^{ref}에 Projection된 p_i의 거리가 최소가 되는 q를 찾는다.

$$\min_{q} \sum_{i} \left\| p_i \oplus q - \Pi\left\{ \mathcal{S}^{\text{ref}}, p_i \oplus q \right\} \right\|^2$$

<FIG.2 Least Square Equation>

- ICP는 반복적으로 위의 식을 최소로 만든다. S^{ref} 위로의 Projection은 Old Guess q_k로 계산되며 Solution으로 New Guess q_{k+1}를 구한다.

$$\min_{q_{k+1}} \sum_{i} \left\| p_i \oplus q_{k+1} - \Pi\left\{ \mathcal{S}^{\text{ref}}, p_i \oplus q_k \right\} \right\|^2$$

<FIG.3 Discrete Least Square Equation>

- PL-ICP에 경우 Point to Line Metric은 다음과 같다. n_i은 Correspond 된 Line의 법선 벡터

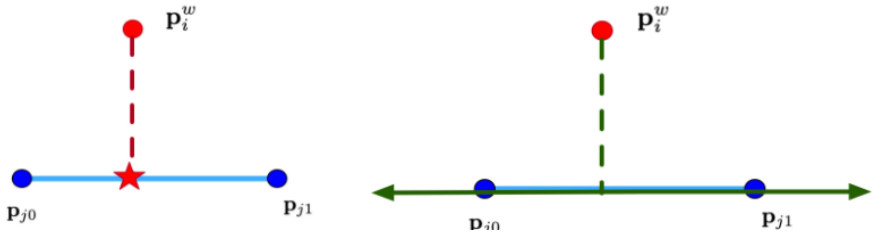
이다.

$$\min_{q_{k+1}} \sum_i \left(n_i^{\mathrm{T}} \left[p_i \oplus q_{k+1} - \Pi\{\mathcal{S}^{\mathrm{ref}}, p_i \oplus q_k\} \right] \right)^2$$

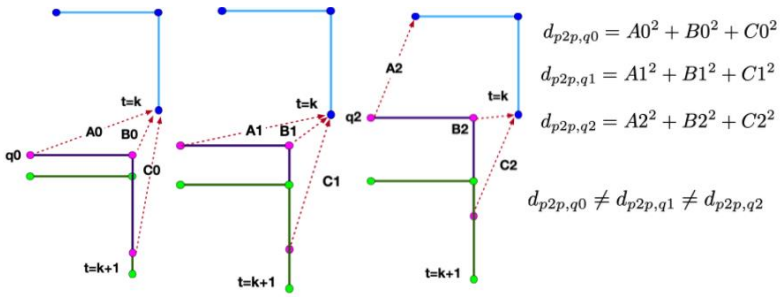
<FIG.4 PL-ICP Metric>

3. Point to Lane[PL-ICP] VS Point to Point ICP[Vanilla-ICP] (reference: PL-ICP, Andrea Censi, 2009)

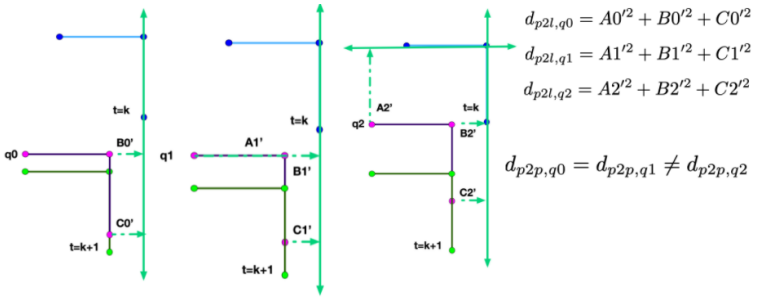
- Vanilla-ICP는 Segment 위에 있는 점에 대한 거리를 비교한다. 하지만 PL-ICP는 Segment를 포함한 Line 위에 있는 점에 대해 거리를 비교한다. <FIG-4>
- Vanilla ICP 경우 q1-q3까지 Score가 모두 다르기 때문에 모든 변환을 거쳐야 하지만 PL-ICP경우 q3까지 Score가 같기 때문에 q1에서 바로 q3로 넘어갈 수 있다. PL-ICP를 사용하면 Space of Transform이 줄어들게 되어 빠르게 수렵하게 된다. <FIG-4,5>
- Vanilla ICP는 Error Metric이 항상 감소하여 Local Minimum에 항상 도달한다. 그리고 일반적으로 Linear Convergence를 보인다.
- PL-ICP에서 Error Metric을 최소화하는 것은 Gauss-Newton iteration과 동일하다. Error Metric을 구하는 알고리즘이 Zero-residual Problem이고 Good first Guess를 가정한다면 Error Metric은 Quadratic Convergence를 보인다
- 정확한 해로 수렴하는 성질이 Vanilla ICP는 Linear인 반면에 PL-ICP는 Quadratic(2차)이므로 PL-ICP 수렴하는 성질은 우수하다
- Reference Point Cloud가 Polyline으로 존재한다면 유한한 number of step으로 최종 해로 수렴하게 된다.



<FIG.5 Vanilla ICP , PL-ICP Searching Correspondence>



<FIG.6 Vanilla-ICP Steps of finding Correspondence>



<FIG.7 PL-ICP Steps of finding Correspondence>

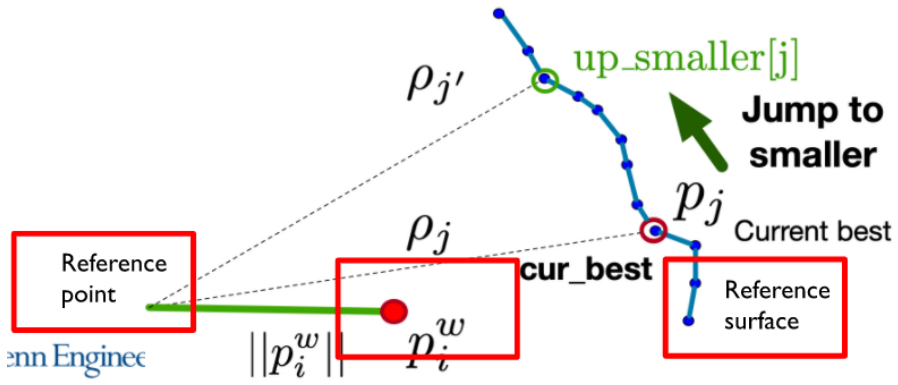
4. Discard Outlier

- Least Square를 이용하여 해를 구하는 과정은 outlier에 매우 취약하므로 사전에 제거해줘야 한다.
- ICP 알고리즘에 라이다 RAW 데이터를 입력하기 전에 Radial Sequence의 Range(거리)값을 분석하여 outlier를 제거해준다.
- 또한 Correspondence를 배열을 만들어 준 뒤 Least Square 식에 넣어 주기 전에 Outlier를 제거해주면 더욱 정확한 Next Guess q를 얻을 수 있다. 대표적인 Outlier 제거 알고리즘에는 RANSAC이 있다.

5. Jump Table

- Besl and McKay(1992)는 ICP 과정에서 전체 running time 중 Correspondence를 찾는 과정이 95%를 차지한다고 발표했다.
- Naïve한 방법으로 Correspondence를 찾게 되면 Correspondence를 찾는 Point Cloud의 한 개의 점에서 Ref Point Cloud 모든 점을 비교해야 하고 이 과정을 Correspondence를 찾는 Point Cloud 모든 점에 대해 수행해야 하므로 Correspondence를 전체적으로 n^2번 수행하게 된다.
- Correspondence를 찾는 Point Cloud의 한 개의 점에서 Correspondence를 찾는 범위를 줄여주는 것이 ICP의 속도를 증가시킬 수 있는 방법 중 하나다. 대표적으로 Jump Table이 있다. Jump table은 Correspondence를 찾는 Point Cloud와 Lidar Sensor의 위치를 기하학적 성질을 이용하여 Ref Point Cloud에서 필수불가결하게 Correspondence 후보가 될 수 없는 점들을 제거해주고 남은 Point Cloud이다.
- 점프테이블은 총 네 개의 변수{up_big, up_small, down_big, down_small}를 사용한다. ref_PC에서의 어떤 점의 순서가 index=j(0<=j<=n, n=sizeof(ref_PC))라고 했을때, j에서 하나씩 증가하면서 pr_k.r (j<k<=n)을 pr_j.r과비교하여 최초로 커질 때 up_big=k , 최초로 작아질 때 up_small=k로 지정한다. 마찬가지로 j에서 하나씩 감소하며(0<=k<j) 같은 방법

으로 비교하여 최초로 커질 때 down_big=k, 최초로 작아질 때 down_small=k라 한다

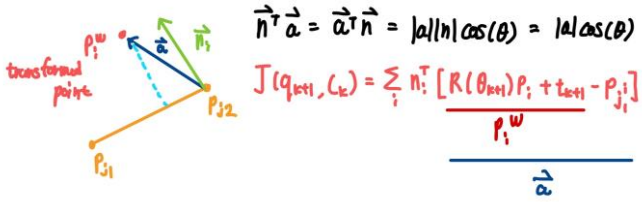


<FIG.8 Jump Table>

6. Least Square(updating transform)

- Formulate a Point to Line error object by finding distance from point to line

$$J(q_{k+1}, C_k) = \sum_i n_i^T [R(\theta_{k+1})P_i + t_{k+1} - P_{ji}]$$



<FIG.9 Point to line Error function>

- Set metric equation that find transform q_{k+1} which minimize the error object.

$$\min_{q_{k+1}} J(q_{k+1}, C_k) = \min_{\theta,t} \sum_i n_i^T [R(\theta_{k+1})P_i + t_{k+1} - P_{ji}] = \min_{\theta,t} \sum_i \|(R(\theta)P_i + t) - \pi_i\|_{C_i}^2 \ , \ \pi_i = P_{ji}$$

- Reduction to quadratic form. A closed-form for the solution can be found. The 3-D solution (t_x, t_y, θ) will be found in the 4-D space x=$[t_x, t_y, \cos\theta, \sin\theta]$ by imposing the constraint $x_3^2 + x_4^2 = 1$

  Given the vector $P_i = (P_{i0}, P_{i1})$, matrix $M_i = \begin{bmatrix} 1 & 0 & P_{i0} & -P_{i1} \\ 0 & 1 & P_{i1} & P_{i0} \end{bmatrix}$

$$\min_x \sum_i \|(R(\theta)P_i + t) - \pi_i\|_{C_i}^2 = \min_x \sum_i (M_i x - \pi_i)^T C_i (M_i x - \pi_i)$$

$$= \min_x \sum_i x^T {M_i}^T C_i M_i x + {\pi_i}^T C_i \pi_i - 2{\pi_i}^T C_i M_i x$$

  Ignoring the constant terms,

$$x^T\left(\sum_x {M_i}^T C_i M_i\right)x + \left(\sum_x -2{\pi_i}^T C_i M_i\right)x \text{ Let, } M=\sum_x {M_i}^T C_i M_i \text{ , } g=\sum_x -2{\pi_i}^T C_i M_i$$

By defining the matrix W=$\begin{bmatrix} 0_{2\times2} & 0_{2\times2} \\ 0_{2\times2} & I_{2\times2} \end{bmatrix}$ and because of the constraint $x_3^2 + x_4^2 = 1$ we can get $x^T W x = 1$

Problem set: find x that $\min_x \sum_i x^T M x + gx,\ x^T W x = 1$

- Formulate Lagrange

$$L(x,\lambda) = x^T M x + gx + (x^T W x - 1),\ \frac{\partial L}{\partial x} = 2x^T M + g^T + 2\lambda x^T W = 0$$

$$x=(2M + 2\lambda W)^{-T} g$$

need to find λ using $x^T W x = 1$

$$g^T(2M + 2\lambda W)^{-1} W (2M + 2\lambda W)^{-T} g = 1, \text{ let } 2M + 2\lambda W = \begin{bmatrix} A & B \\ B^T & D + 2\lambda I \end{bmatrix}$$

$$(2M + 2\lambda W)^{-1} = \begin{bmatrix} A & B \\ B^T & D + 2\lambda I \end{bmatrix}^{-1} = \begin{bmatrix} * & -ABQ^{-1} \\ * & Q^{-1} \end{bmatrix} =, \text{where } Q=(D-B^T A^{-1} B + 2\lambda I) = (S + 2\lambda I)$$

Let p(λ)=det(S+2λI), we finally get the λ from

$$\lambda^2 4g^t \begin{bmatrix} A^{-1}BB^TA^{-T} & A^{-1}B \\ (symm) & I \end{bmatrix} + \lambda 4 g^t \begin{bmatrix} A^{-1}BS^AB^TA^{-T} & -A^{-1}BS^A \\ (symm) & S^A \end{bmatrix} + g^t \begin{bmatrix} A^{-1}BS^{A^T}S^AB^TA^{-T} & -A^{-1}BS^{A^T}S^A \\ (symm) & S^{A^T}S^A \end{bmatrix}$$
$$= \mathrm{p}(\lambda)^2$$

3. Contribution: Smart Correspondence

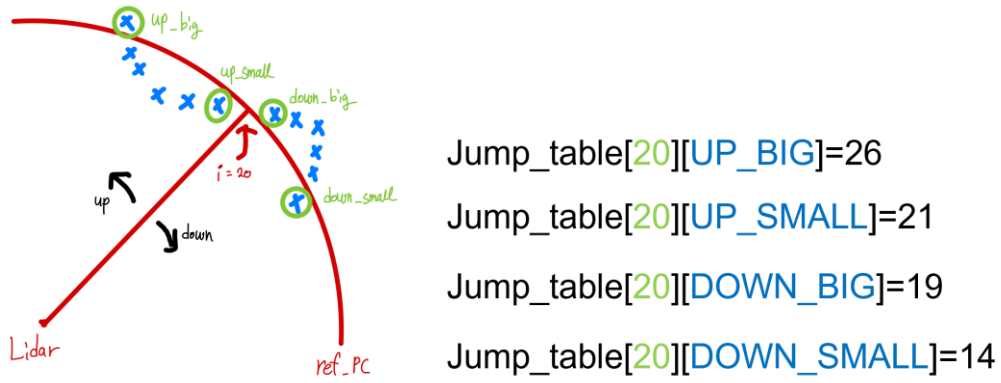
1. Jump_table

- 점프테이블은 총 네 개의 변수{UP_BIG, UP_SMALL, DOWN_BIG, DOWN_SMALL}를 사용한다. ref_PC에서의 어떤 점의 순서가 index=j($0 \le j \le n$, n=sizeof(ref_PC))라고 했을때, j에서 하나씩 증가하면서 P_k^{ref}.r ($j < k \le n$)을 P_j^{ref}.r과 비교하여 최초로 커질 때 UP_BIG=k, 최초로 작아질 때 UP_SMALL=k로 지정한다. 마찬가지로 j에서 하나씩 감소하며 (0≤k＜j) 같은 방법으로 비교하여 최초로 커질 때 DOWN_BIG=k, 최초로 작아질 때 DOWN_SMALL=k라 한다

$P_j^{ref}(0 \le j \le 1079)$
- UP_BIG: for(j≤k≤1079) P_j^{ref}.r< P_k^{ref}.r break;
- UP_SMALL: for(j≤k≤1079) P_j^{ref}.r> P_k^{ref}.r break;
- DOWN_BIG: for(0≤k≤j) P_j^{ref}.r< P_k^{ref}.r break;
- DOWN_SMALL: for(0≤k≤ j) P_j^{ref}.r> P_k^{ref}.r break;

<FIG.10 Setting a Jump table>

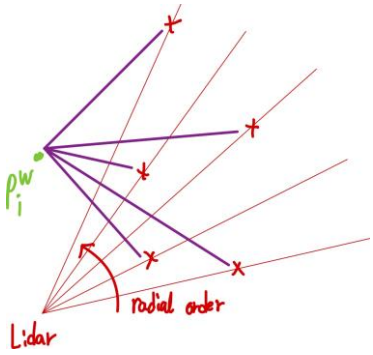
- 예를 들어 현재 Reference PC의 20번째 index의 Jump_table을 결정한다고 할 때 up방향(index 증가 방향)으로 처음으로 작아지는 index를 UP_SMALL 커지는 index를 UP_BIG이라고 한다. Down 방향도 마찬가지이다. 결국 Reference PC 각각의 점들에 대해 Jump_table index가 4개씩 저장된다.



<FIG.11 A example of Jump table>

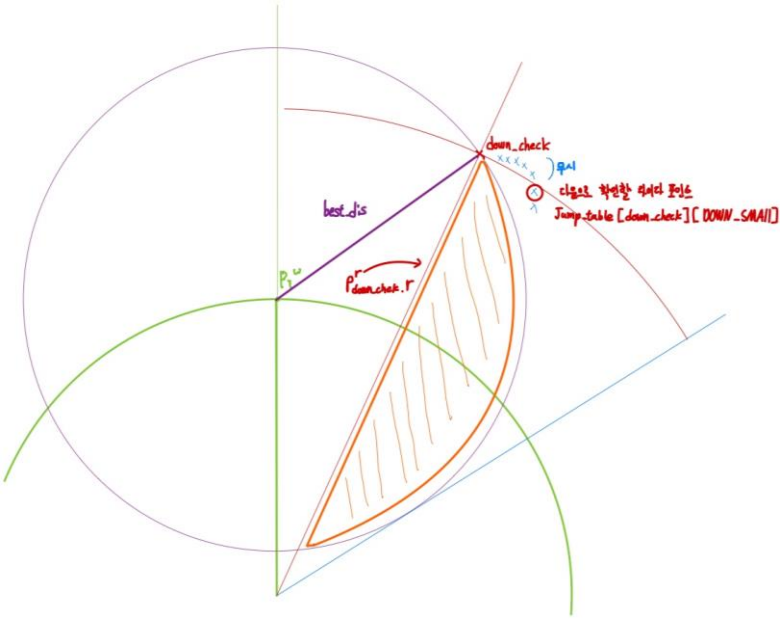
2 문제점 해결

- Correspondence를 찾는 것은 Ref PC의 점들 중 현재 정합 하려는 PC의 한 점 P_i^w와 가장 가까운 점을 찾는 것이다. Radial order으로 표현되어 있는 Ref PC의 점들과 거리를 비교하며(보라색 선) 가장 짧은 선을 찾고 해당하는 Ref PC index를 P_i^w의 best_point로 저장한다.



<FIG.12 Finding Corresponding index>

- Down 방향으로 탐색한다고 가정하면 현재 탐색하고 있는 라이다 index(down_check)와 P_i^w의 거리가 best_dis보다 짧기 위해서는 주확생 영역에 P_j^{ref}가 존재해야 한다. P_j^{ref}.r를 기준으로 Jump_table 정한 것을 참고 할 때 직관적으로 보면 $P_{down\_check}^{ref}$.r보다 긴 index들은, 다시 말해서 Jump_table을 정할 때 BIG으로 판별된 index들은, 주황색 영역에 들어갈 수 없음을 알 수 있다. 그러므로 down_check에서 index를 한 개씩 줄어가며 P_i^w의 거리를 계산하여 best_dis와 비교하는 것보다 DOWN_SMALL로 바로 이동하여 비교하는 것이 더욱 효율적이다.



<FIG.13 Searching down direction>

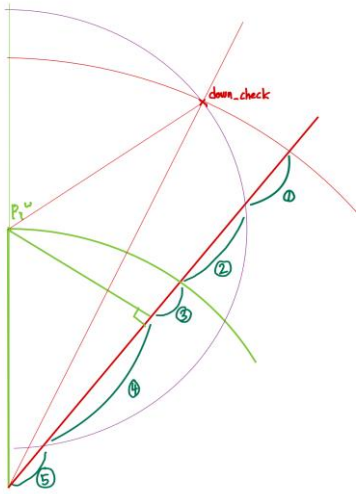
- 현재 탐색하고 있는 Index에서 다음으로 탐색할 index를 BIG 또는 SMALL로 정할지에 대한 기준을 Censi 논문에서는 다음과 같이 설명했다.

Checking Direction: UP IF(P_i^w.r $> P_{up\_check}^{ref}$.r) next_check=Jump_table[up_check][UP_BIG]
IF(P_i^w.r $< P_{up\_check}^{ref}$.r) next_check=Jump_table[up_check][UP_SMALL]

Checking Direction: DOWN IF(P_i^w.r $> P_{down\_check}^{ref}$.r) next_check=Jump_table[down_check][DOWN_BIG]
IF(P_i^w.r $< P_{down\_check}^{ref}$.r) next_check=Jump_table[down_check][DOWN_SMALL]

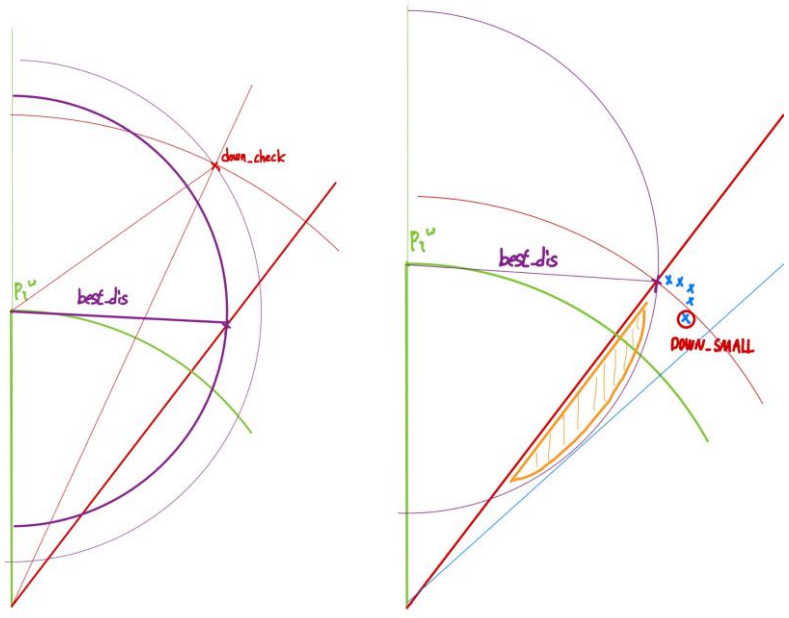
<FIG.14 Criterion of selecting which index we are going to search>

- 위 사진에서 Jump_table[down_check][DOWN_SMALL] index가 위치할 수 있는 곳은 다음과 같이 5가지로 나눌 수 있다. 만일 1번 구역에 위치한다면 P_i^w부터의 거리가 best_dis보다 크게 되어 best_dis는 업데이트 되지 않고 해당 index의 DOWN_SMALL index로 이동하여 다시 거리를 비교하게 된다.



<FIG.15 Five candidates that the Ref point should locate>

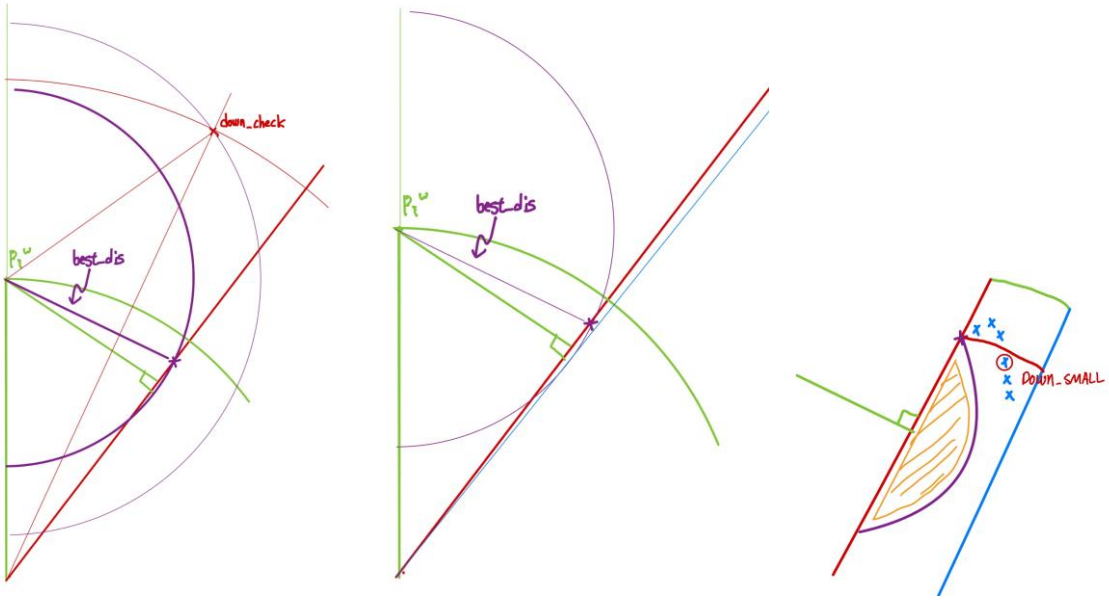
- 만일 Jump_table[down_check][DOWN_SMALL] index가 2번 구역에 위치할 경우 best_dis는 업데이트 되고 새로운 원이 그려진다. 계속해서 down 방향으로 탐색하면서 best_dis를 업데이트 하기 위해서 P_j^{ref}가 주황색 영역에 존재해야 한다. 현재 $P_i^w.r < P_{down_check}^{ref}.r$를 만족하므로 해당 index의 DOWN_SMALL로 이동한다.



<FIG.16 What if the Ref point locate at zone 2 >

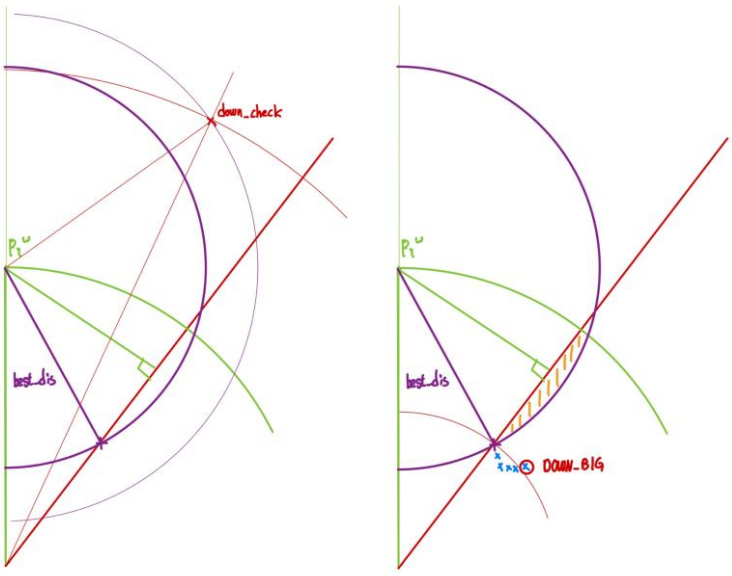
- 문제는 Jump_table[down_check][DOWN_SMALL] index가 3번 구역에 위치할 경우에 일어난다. 위의 경우와 마찬가지로 best_dis가 업데이트 되어 새로운 원이 그려진다. 계속해서 down 방향으로 탐색하면서 best_dis가 업데이트 되기 위해서 P_j^{ref}가 확대한 그림에서 주황색 영역에 존재해야 한다. Censi가 설정한 조건에서는 다음으로 탐색할 index를 정한다면 $P_i^w.r > P_{down_check}^{ref}.r$이므로 DOWN_BIG이지만 그림에서 확인할 수 있듯이 DOWN_BIG

에 해당하는 index는 주황색 영역에 위치할 수 없다. 해당 상황에서는 다음으로 탐색할 index가 DOWN_SMALL이 되어야 한다.



<FIG.17 What if the Ref point locate at zone 3; A problem Occurs !! >

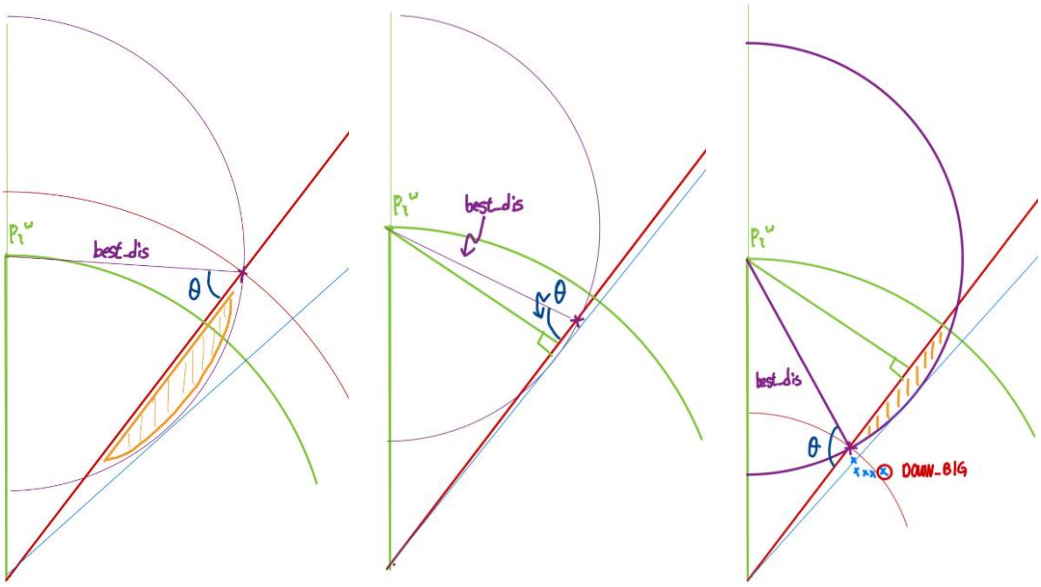
- 다음은 Jump_table[down_check][DOWN_SMALL] index가 4번 구역에 위치할 경우이다. 마찬가지로 best_dis가 업데이트 되어 새로운 원이 그려진다. 계속해서 down 방향으로 탐색하면서 best_dis를 업데이트 하기 위해서 P_j^{ref} 가 주황색 영역에 존재해야 한다. 현재 P_i^w.r > $P_{down\_check}^{ref}$.r를 만족하므로 해당 index의 DOWN_SMALL로 이동해야 하고 이는 타당한 것을 그림을 통해서 알 수 있다.



<FIG.18 What if the Ref point locate at zone 4 >

- 현재 index에서 다음으로 탐색할 index를 BIG 또는 SMALL로 정할지에 대한 기준을

Censi 논문에서 제시한 것의 문제점을 해결한 새로운 방법론을 발견했다. best_dis가 update되어 새로운 원이 그려졌을 때 해당 index에서 P_i^w와 라이다 원점 사이의 각도를 이용한다. 각도가 예각이면 다음으로 SMALL index로 이동하고 둔각이면 BIG index로 이동한다. 다르게 표현하면, 원이 새롭게 만들어 졌을 때 생긴 활꼴의 위치가 down_check 보다 위에 있으면 BIG index로 이동하고 아래에 있으면 SMALL index로 이동한다. 각 후보에 대한 다음 탐색 인덱스를 아래와 같이 비교할 수 있다.

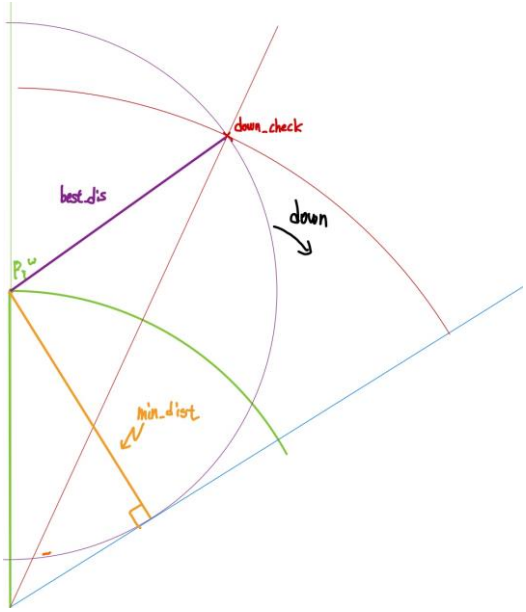


<FIG.19 New criterion of selecting which index we are going to search >

			Censi's	Ours
후보 ①	θ: 예각, $P_i^w.r > P_{down\_check}^{ref}.r$	best_dis not updated	DOWN_SMALL	DOWN_SMALL
후보 ②	θ: 예각, $P_i^w.r > P_{down\_check}^{ref}.r$	best_dis updated	DOWN_SMALL	DOWN_SMALL
후보 ③	θ: 예각, $P_i^w.r < P_{down\_check}^{ref}.r$	best_dis updated	DOWN_BIG	DOWN_SMALL
후보 ④	θ: 예각, $P_i^w.r < P_{down\_check}^{ref}.r$	best_dis updated	DOWN_BIG	DOWN_BIG
후보 ⑤	θ: 둔각, $P_i^w.r < P_{down\_check}^{ref}.r$	best_dis not updated	DOWN_BIG	DOWN_BIG

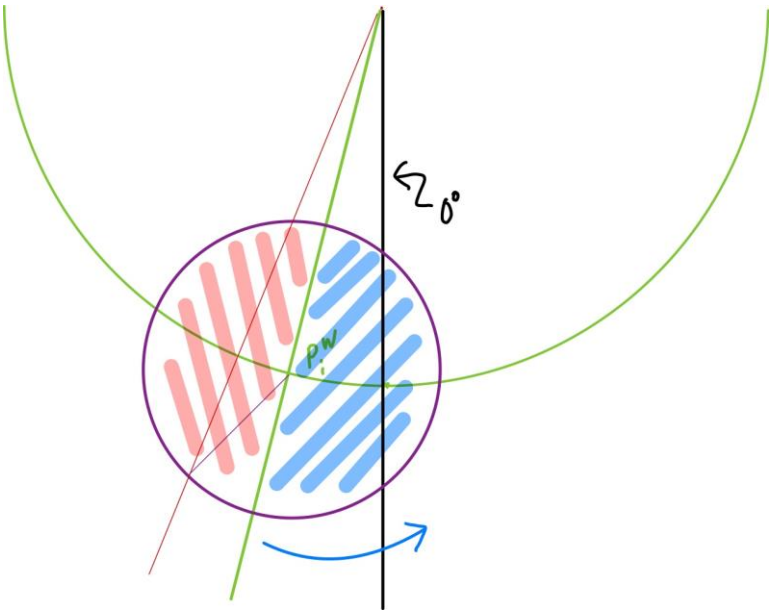
<FIG.20 Compare our new criterion with others>

- 한 쪽 방향의 탐색은 min_dist가 현재 best_dis보다 크게 되면 중지하고 시작점으로 돌아가 반대 방향으로 탐색을 시작한다. min_dist는 P_i^w 현재 탐색하고 있는 Ref PC index의 원점으로부터 방사 vector에 수직으로 내린 선의 길이이다. 두 방향으로의 탐색이 끝나면 현재 best_point를 Correspondence Vector에 P_i^w 와 함께 저장하고 P_{i+1}^w 에 대해 Correspondence를 다시 시작한다.



<FIG.21 End up searching in one direction>

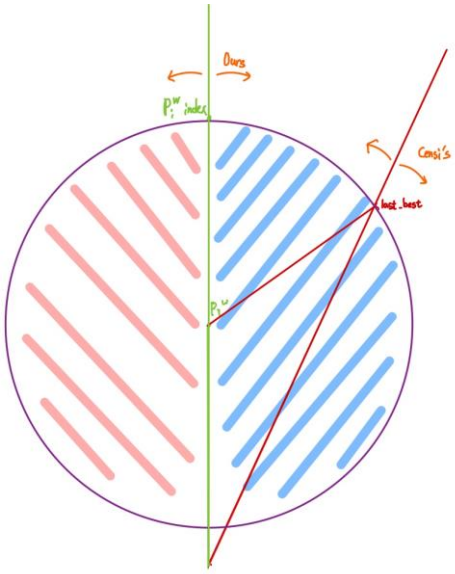
- Censi 논문에서 제시한 알고리즘의 또 다른 작은 문제는 라이다 스캔 범위가 360도를 넘어갈 때 발생한다. 아래 그림에서 Censi는 360도로 증가하다 0도로 넘어가 다시 up 방향으로 탐색을 하지 않고 360도에 탐색을 중지한다. 그래서 우리는 탐색하는 도중 0도 또는 360도를 마주쳤을 때 넘겨주는 알고리즘을 추가했다. 그리고 jump table의 개념은 up 방향으로는 파란색 반원, down 방향으로는 빨간색 반원에만 적용된다. 그래서 두 방향의 최대 탐색 index를 start_index를 P_i^w방향의 index라 했을 때 원 반대편에 위치한 index로 설정한다.



3. 속도 개선

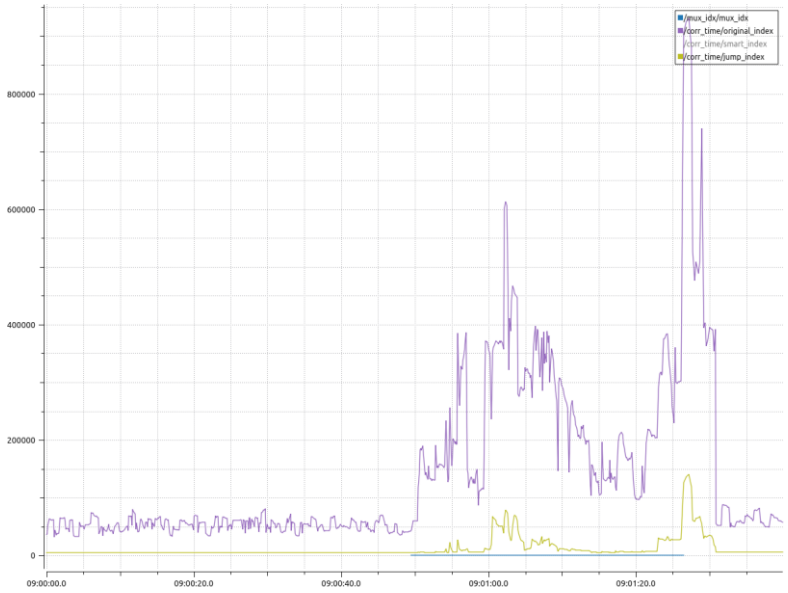
- 탐색을 시작하는 Start_index 변경하여 Correspondence 전체 시간을 단축했다. Censi 논

문에서는 start_index를 last_best 부근으로 설정했다. 위에서 설명한 것과 같이 jump table 알고리즘은 P_i^w방향에 대해서 반원에만 적용되기 때문에 아래 그림에서 down 방향으로는 바로 jump table을 사용할 수 있지만 up 방향으로는 P_i^w방향까지 도달해야 사용할 수 있다. 그래서 우리는 처음부터 양 방향 모두 jump table을 사용할 수 있도록 start_index를 P_i^w방향에 위치한 index로 설정했다.



4. 결과

- F1TENTH simulator를 이용하여 Censi's Jump Table과 Ours Jump Table을 비교했다. 차량을 이동하며 두개의 다른 pointcloud를 똑같이 Censi's Jump Table과 Ours Jump Table에 입력했다. 결과를 오픈소스 Plotjuggler를 활용하여 표현했다.
- 아래 그래프에서는 Jump table을 사용했을 때, 정합 하려는 pointcloud 전체 1080개 점에 대한 Correspondence 단계에서 탐색하는 index의 총 합을 표현했다. 보라색은 Censi's Jump Table이며 노란색 선은 Ours Jump Table의 결과이다. Start_index를 바꿔 줌으로써 시간이 탐색하는 index의 수가 현저히 줄어든 것을 확인할 수 있다.



- 아래 그래프에서는 Jump table을 사용했을 때, Correspondence에 걸리는 시간을 비교했다. 초록색 선은 Censi's Jump Table이고 분홍색 선은 Ours Jump Table의 결과이다. 시간 측면에서 개선된 것을 확인할 수 있다.