**경로생성**

**PathAttacher**

**알고리즘 설명서**

**기차 프로젝트**

최종 작성일: 2019년 02월19일

PathAttacher

[PathAttacher 2](#_Toc1500597)

[1 목적 3](#_Toc1500598)

[2 개요 4](#_Toc1500599)

[3 알고리즘 6](#_Toc1500600)

[6](#_Toc1500601)

## 목적

앞 카트의 자세와 경로를 받아 추가할 경로를 반환한다

## 개요

끝 자세와 앞 카트로부터 새로 받은 자세 사이의 거리가

**distance >distance\_tol :**  dubinspath로 연결하고

**distance\_tol > distance > distance\_max :** 일직선으로 연결하고

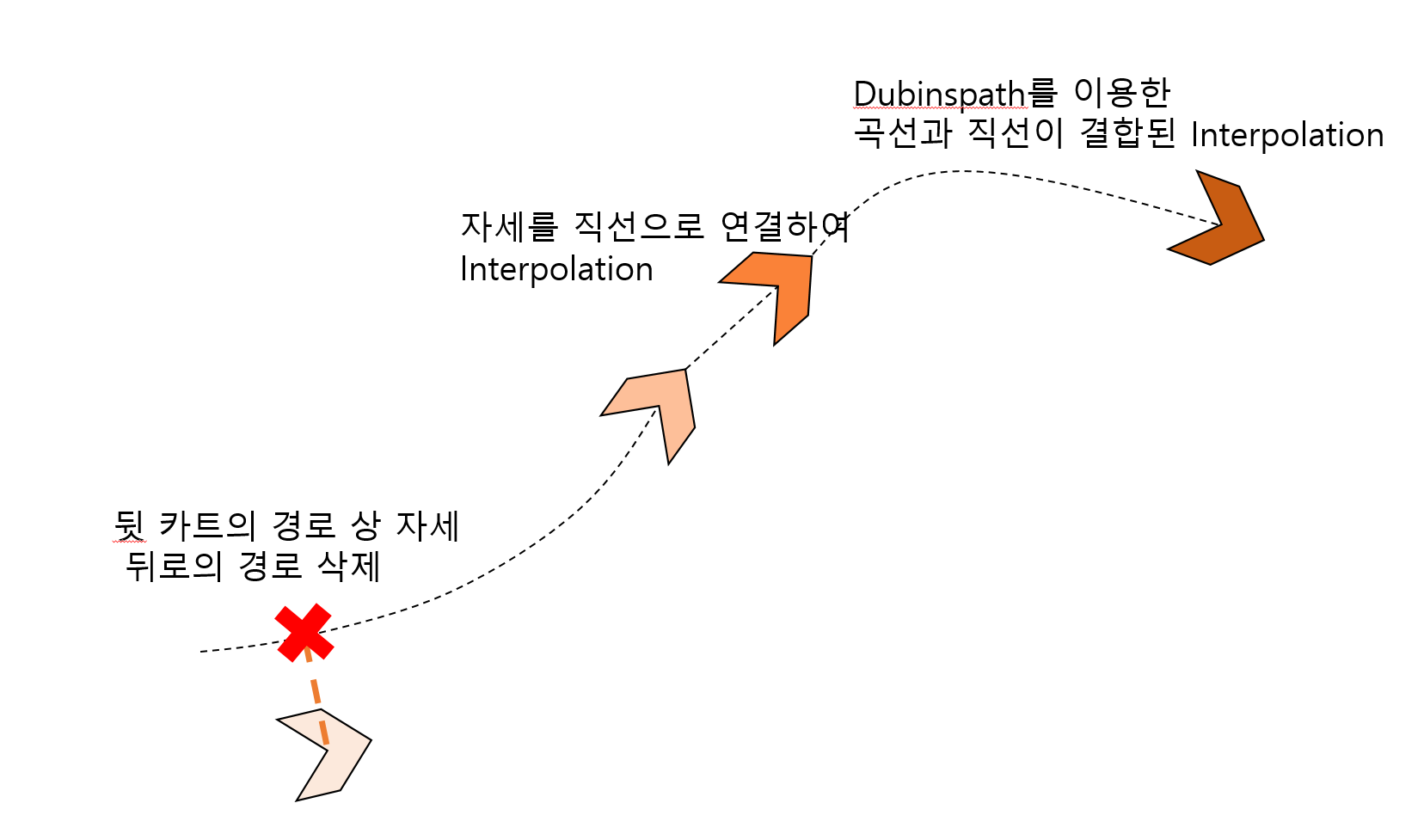
**distance\_max > distance > distance\_min :**

끝 자세와 새로 받은 자세를 첫, 그리고 끝 자세로 갖는 Segment를 경로에 추가

**distance\_min > distance :** 새로 받은 자세를 연결하지 않는다

|  |
| --- |
| **Algorithm 2. Linearpath Algorithm** |
| **INPUT: Cpose frontCartPose, const Path \*trainPath**  **OUTPUT**: Path |
| **double** travel\_range = 0.0;  poseStore[0]= q0[0];  poseStore[1]= q0[1];  poseStore[2]= q0[2];  **while**( travel\_range < distance )  {  **CPose** startPose;  **CPose** endPose;  startPose.setData(poseStore[0], poseStore[1], poseStore[2]);  **CPathSegment** newSegment;  newSegment.setStartPose(startPose);  poseStore[0] =( q1[0]- q0[0] ) \* (travel\_range / distance) + q0[0];  poseStore[1] =( q1[1]- q0[1] ) \* (travel\_range / distance) + q0[1];  poseStore[2] =( q1[2]- q0[2] ) \* (travel\_range / distance) + q0[2];  endPose.setData(poseStore[0],poseStore[1], poseStore[2]);  newSegment.setEndPose(endPose);  newPath.push\_back(newSegment);  travel\_range += m\_StepSize;  }  ROS\_INFO("Linear"); |
| *StepSize 의 간격으로 직선 사이의 자세를 잇습니다.* |

## 알고리즘



1) Dubinspath

직선과 곡선을 결합하여 자세들을 연결한다.

참조: <http://github.com/AndrewWalker/Dubins-Curves>

2) Linearpath

자세들을 직선으로 연결하여 m\_StepSize(Interpolation할 자세 간의 거리)의 간격으로 Interpolation

첫 자세와 끝 자세 간의 거리 중에 Interpolation한 자세까지의 거리 만큼(travel\_range / distance)을 첫 자세와 끝 자세의 x, y 좌표 차이에 곱해준 후 다시 travel\_range += m\_StepSize