Outlier removal algorithm in pose transformation

Zhitao Wu

Hangzhou Dianzi University

Abstract: In SLAM algorithms, each keyframe is typically treated as a node, and sensor data is used to compute the pose transformations (rotation matrix R and translation vector t) between different keyframes. These pose transformations can be viewed as edges connecting the nodes, allowing the relationships between the nodes to be represented in the form of a graph. However, in this graph, some pose transformations may be outliers, causing the corresponding edges to be inconsistent with the other edges in the graph. We propose an algorithm to detect pose transformations that are outliers within the graph and eliminate these outliers.

Keywords: transformation, outlier.

1. Introduction

Paper [1] proposed a dynamic covariance scaling method to address the outlier problem in graph optimization. Paper [2] introduced a robust estimation approach based on progressive non-convex optimization, specifically designed for handling outliers in the context of pose graph optimization.

2. Problem formulation

In SLAM algorithms, each keyframe is typically treated as a node, and sensor data is used to compute the relative pose transformations (rotation matrix R and translation vector t) between different keyframes. These pose transformations are analogous to edges connecting the nodes, allowing the relationships between nodes to be represented in the form of a graph. However, in this graph, certain pose transformations may contain outliers, causing the corresponding edges to be inconsistent with other edges in the graph.

The same scenario also occurs in sensor networks. In the sensor network localization problem, each sensor represents a node, and the sensors can measure the relative pose transformations between themselves and other nodes. This sensor network can therefore be represented in the form of a graph, where the nodes of the graph correspond to the sensors, and the edges represent the pose transformations between the sensors. Due to factors such as noise, the directly measured pose transformations between different nodes may include outliers, leading to inconsistencies between certain edges and the rest of the graph. To address this issue, we propose an algorithm to eliminate outlier pose transformations in the graph, or equivalently, those inconsistent edges.

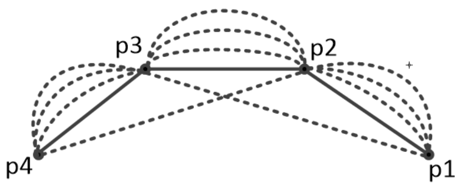

Fig1. node location and measurement examples

Figure 1 illustrates an example of the problem. In this figure, there are four nodes in total, and a dashed line between any two nodes represents a relative pose transformation obtained from a single measurement. The pose transformation is represented using a homogeneous transformation matrix $T \in R^{4\times4}$. Between Node 1 and Node 2, there are three measurements, yielding the relative poses $T_{12}^1, T_{12}^2, T_{13}^3$. Similarly, there are three measurements between Node 2 and Node 3, as well as between Node 3 and Node 4. Additionally, for non-adjacent nodes, one measurement exists between Node 1 and Node 3, resulting in the relative pose transformation T_{13}, and one measurement between Node 2 and Node 4, resulting in the relative pose transformation T_{24}. We then assume that among the measurements between Node 3 and Node 4, one of the measurements, specifically T_{34}^3, is an outlier. The goal of the algorithm is to identify and remove this outlier.

3. Algorithm procedure

Let p_1 and p_2 be two nodes in the graph. Three measurements were performed between p_1 and p_2, resulting in three relative pose transformation matrices, $T_{12}^1, T_{12}^2, T_{12}^3$.

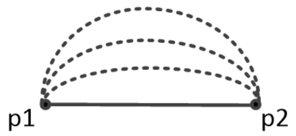

Fig2. measurement without using intermediate nodes

At the same time, p_1 and p_2 can also be connected via another intermediate node (e.g., p_3, p_4, p_5), allowing the relative pose transformation between p_1 and p_2 to be represented through the intermediate nodes as follows:

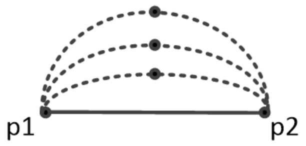

Fig3. measurement with the help of one intermediate node

$$T_{12}^{4} = T_{13}T_{32}$$

$$T_{12}^{5} = T_{14}T_{42}$$

$$T_{12}^{6} = T_{15}T_{52}$$

In a similar manner, p_1 and p_2 can also be connected through another two intermediate nodes (e.g., p_6 and p_7, p_8 and p_9, or p_{10} and p_{11}). In this way, the pose transformation between p_1 and p_2 can be represented with the aid of two intermediate nodes.

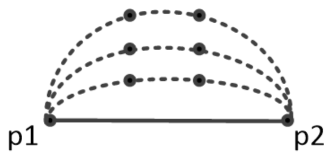

Fig4. measurement with the help of two intermediate nodes

$$T_{12}^{7} = T_{16}T_{67}T_{72}$$

$$T_{12}^{8} = T_{18}T_{89}T_{92}$$

$$T_{12}^{9} = T_{1,10}T_{10,11}T_{11,2}$$

By analogy, the pose transformation between p_1 and p_2 can be represented through multiple intermediate nodes. The identification of these intermediate nodes can be achieved using various path planning algorithms (such as the A* algorithm). Suppose that through this method, a total of n pose transformations between p_1 and p_2, denoted as $T_{12}^{i}, i \in [1, n]$ are obtained.

If the pose transformations between all the nodes involved in the calculation of the pose transformation between p_1 and p_2 are only subject to common noise (e.g., Gaussian noise with small variance) and no outlier noise is present, then the obtained $T_{12}^{i}, i \in [1, n]$ should be relatively consistent. Conversely, if outlier noise exists in the pose transformations between certain nodes, some of the values in $T_{12}^{i}, i \in [1, n]$ will exhibit outliers.

We decompose the pose transformation matrix T_{12}^{i} into a rotation matrix R_{12}^{i} and a translation vector t_{12}^{i}, and then remove outliers from the rotation matrices R_{12}^{i} and translation vectors t_{12}^{i} separately. If, for a pose transformation matrix T_{12}^{i} where $i = i_1$, either the corresponding rotation matrix R_{12}^{i1} or the translation vector t_{12}^{i1} contains an outlier, the entire pose transformation matrix T_{12}^{i1} is considered an outlier. The detailed steps for removing outliers from the rotation matrices $R_{12}^{i}, i \in [1, n]$ and translation vectors $t_{12}^{i}, i \in [1, n]$ are described in subsequent sections.

It is known that there are a total of n relative pose transformations, denoted as $T_{12}^{i}, \in [1, n]$, between points p_1 and p_2. Using an outlier removal algorithm, we identify that a specific pose transformation T_{12}^{i}, where $i = i1$, is an outlier. If T_{12}^{i1} passes through only a single edge (without involving other nodes), then the edge corresponding to T_{12}^{i1} is considered an outlier with a probability $p = 1$. If T_{12}^{i1} passes through two edges (e.g., $T_{12}^{4} = T_{13}T_{32}$), we cannot determine whether the outlier lies specifically in T_{13} or T_{32}, or if both T_{13} and T_{32} contain outliers. In this case, the probability of each edge being an outlier can be set as $p = 0.5$. If T_{12}^{i1} passes through three edges (e.g., $T_{12}^{7} = T_{16}T_{67}T_{72}$), we cannot determine whether the outlier lies specifically in T_{16}, T_{67}, or T_{72}, or whether all of them are outliers. In this case, the probability of each edge being an outlier can be set as $p = 1/3$. This process can be generalized further in a similar manner.

The input data is typically in the form of a graph $G = [V, E]$, where the nodes V represent the keyframes in SLAM or the nodes in a sensor network, and the edges E represent the pose transformations between different nodes. Using this approach, we can compute the involved pose transformations $T_{k,k+1}^{i}$, where $i \in [1, n]$, for any two adjacent nodes p_k and p_{k+1} in the graph. Subsequently, we identify the pose transformations $T_{k,k+1}^{i}$, where $i \in S$, that belong to outliers, and iteratively accumulate the probabilities p of the edges being outliers.

If the number of paths found between two adjacent nodes p_k and p_{k+1} by a path-planning algorithm (e.g., the A* algorithm) is too small, resulting in an insufficient number of feasible pose transformations $T_{k,k+1}^{i}, i \in [1, n_1]$ (where $n_1 < n_{thres}$), then outlier detection may be omitted. This is because the small number of data points would render the results of outlier detection unreliable.

We can identify outliers for any two adjacent nodes $p_k, p_{k+1} \in V$ in the graph $G = [V, E]$ using the aforementioned steps and accumulate the probability p_i of each edge $e_i \in E$ being an outlier. After evaluating all pairs of adjacent nodes in the graph, edges can be removed based on their probability of being outliers. For example, if the probability p_i for edge e_i exceeds the threshold p_{thres} (which can be set to 1 or a smaller value), the edge eie_iei can be classified as an outlier and removed. After completing these steps, the remaining pose transformations (edges in the graph) in $G = [V, E]$ will be more reliable. Using these remaining pose transformations, more accurate node coordinates can be computed, leveraging algorithms from SLAM or sensor network localization. The pseudocode of the algorithm is as follows.

Outlier removal algorithm in pose transformation

```
Input: A graph $G = [V, E], e_i \in E$ represents a pose transformation

Output: A graph $G' = [V, E']$ with outliers removed

For (any two adjacent points $p_k, p_{k+1} \in V$):

    Use a path planning algorithm to attempt to find $n_{expect}$
    paths from $p_k$ to $p_{k+1}$, and let n be the number of paths
    actually found.
    If ($n < n_{thres}$):
        break
    For the n paths from $p_k$ to $p_{k+1}$, calculate the pose
    transformations $T^i_{k,k+1}, i \in [1, n]$.
    Use an outlier removal algorithm to identify the outliers
    among $T^i_{k,k+1}, i \in [1, n]$, denoted as $T^i_{k,k+1}, i \in S$.
    For ($i \in S$):
        Assume that $T^i_{k,k+1}$ is obtained by multiplying the pose
        transformations of m edges $e_j$, where $j \in [1, m]$.
        Let $p_j$ be the probability that edge $e_j$ is an outlier.
        Update the probability as $p_j = p_j + \frac{1}{m}, j \in [1, m]$.
For ($e_i \in E$):
    If $p_i > p_{thres}$
        The pose transformation associated with $e_i$ contains
        outliers, and $e_i$ is removed from E.
The resulting graph with outliers removed is $G' = [V, E']$.
```

4. Outlier removal algorithm

4.1. Outlier removal algorithm with weighted IQR

There are many outlier removal algorithms, such as the density-based outlier removal algorithm (DBSCAN), the clustering-based outlier removal algorithm (K-means), and the distribution-based outlier removal algorithm (IQR), among others. In the current problem, we assume that the data points have only one central cluster. To reduce computational complexity, we choose the IQR (Interquartile Range) algorithm for outlier removal. To better adapt the IQR algorithm to the current problem, we propose an improved version—a weighted IQR algorithm.

We begin by briefly introducing the process of the IQR algorithm. Let the input data be $X = \{x_i \in R, i \in [1, n]\}$, which consists of n data points x_i, where $i \in [1, n]$, and the data points are sorted in ascending order. The next step involves calculating the first quartile (Q1), the third quartile (Q3), and the interquartile range (IQR). The first quartile (Q1) is computed as follows. When the number of data points n is odd, Q1 corresponds to the value at the position $0.25 \times (n + 1)$ (this can be determined using interpolation if necessary). When n is even, Q1 is the interpolated value between the data points at positions $0.25 \times n$ and $0.25 \times (n + 1)$. The third quartile (Q3) is calculated in a similar manner: For an odd number of data points, Q3 corresponds to the value at the position $0.75 \times (n + 1)$. For an even number of data points, Q3 is the interpolated value between the data points at positions $0.75 \times n$ and $0.75 \times (n + 1)$. The interquartile range (IQR) is then defined as the difference between Q3 and Q1.

$$IQR = Q3 - Q1$$

Then, the outlier boundaries are calculated based on the Interquartile Range (IQR). Typically, a value of $k = 1.5$ is used to define the threshold for identifying outliers. The lower bound for outliers is computed using the formula $Q1 - 1.5 \times IQR$, while the upper bound for outliers is determined using the formula $Q3 + 1.5 \times IQR$. These two limits establish a reasonable range for the data, with any data points lying outside this range considered as outliers.

Based on the calculated upper and lower bounds, data points in the dataset that are less than the lower bound or greater than the upper bound are marked as outliers. Specifically, the criteria are as follows: if $x_i < Q1 - 1.5 \times IQR$, then x_i is considered a lower outlier; if $x_i > Q3 + 1.5 \times IQR$, then x_i is considered an upper outlier. The dataset can be filtered by retaining only the data points within the range $[Q1 - 1.5 \times IQR, Q3 + 1.5 \times IQR]$.

The advantage of the IQR algorithm lies in its use of quantile calculations, which makes it insensitive to outliers and computationally efficient. When applying the IQR algorithm to the current problem, the data points are denoted as $R^i_{12}, i \in [1, n]$. Some of these R^i_{12} are directly obtained through measurements, while others are derived by multiplying multiple pose transformations. As a result, the reliability of different R^i_{12} values varies. To address this, we propose a weighted IQR algorithm.

Let the input data be $X = \{x_i \in R, i \in [1, n]\}$, which consists of nnn data points $x_i, i \in [1, n]$, sorted in ascending order. The corresponding weight set for the data points is denoted as $W = \{w_i \in R, i \in [1, n]\}$. Here, the weight w_i represents the reliability of the data point x_i, or the probability that it is not an outlier. This weight is typically determined by the variance of the sensor that measures x_i; the smaller the variance of the sensor, the higher the reliability of the data x_i. Therefore, when calculating the primary data distribution interval (i.e., the IQR interval), more weight should be assigned to data points with higher reliability.

Compared to the standard IQR algorithm, the weighted IQR algorithm requires the following modifications in the steps for calculating Q1 and Q2. We need to compute a cumulative weight set, $W_{add} = \{w^{add}_i \in R, i \in [1, n]\}$, where each element w^{add}_i is calculated as follows.

$$w^{add}_i = \sum_{j=1}^{i} w_i$$

The total weight sum, W_{sum}, is computed as follows.

$$W_{sum} = \sum_{j=1}^{n} w_i$$

The calculation of the first quartile (Q1) is as follows. In the cumulative weight set W_{add}, find a weight w^{add}_{i1} such that it is equal to $0.25 \times W_{sum}$ ($w^{add}_{i1} = 0.25 \times W_{sum}$). In this case, $Q1 = x_{i1}$. Alternatively, if no exact match is found, identify the two weights w^{add}_{i1} and w^{add}_{i1+1} in W_{add} that are closest to $0.25 \times W_{sum}$, satisfying $w^{add}_{i1} < 0.25 \times W_{add} < w^{add}_{i1+1}$. In this case, $Q1 = (x_{i1} + x_{i1+1})/2$. Similarly, the calculation of the third quartile (Q3) follows the same approach. In the cumulative weight set W_{add}, find a weight w^{add}_{i2} such that it is equal to $0.75 \times W_{sum}$ ($w_{i2}^{add} = 0.75 \times W_{sum}$). In this case, $Q3 = x_{i2}$. Alternatively, if no exact match is found, identify the two weights w_{i2}^{add} and w_{i2+1}^{add} in W_{add} that are closest to $0.75 \times W_{sum}$, satisfying $w_{i2}^{add} < 0.75 \times W_{add} < w_{i2+1}^{add}$. In this case, $Q3 = (x_{i2} + x_{i2+1})/2$. The weighted IQR algorithm follows the same steps as the standard IQR algorithm in other aspects. The pseudo-code for the weighted IQR algorithm is as follows.

Weighted IQR algorithm

```
Input: Data set $X = \{x_i \in R, i \in [1, n]\}$, Weight set $W = \{w_i \in R, i \in [1, n]\}$

Output: Filtered data set $X' = \{x'_i \in R, i \in [1, m]\}$

Cumulative weight set
$W_{add} = \{w_i^{add} \in R, i \in [1, n]\}, w_i^{add} = \sum_{j=1}^{i} w_i$.

Total weight sum $W_{sum} = \sum_{j=1}^{n} w_i$.

For ($i \in [1, n]$):
    If($w_i^{add} = 0.25 \times W_{sum}$):
        $Q1 = x_i$.
    If($w_i^{add} < 0.25 \times W_{sum}$ and $w_{i+1}^{add} > 0.25 \times W_{sum}$):
        $Q1 = (x_i + x_{i+1})/2$.
For ($i \in [1, n]$):
    If($w_i^{add} = 0.75 \times W_{sum}$):
        $Q3 = x_i$.
    If($w_i^{add} < 0.75 \times W_{sum}$ and $w_{i+1}^{add} > 0.75 \times W_{sum}$):
        $Q3 = (x_i + x_{i+1})/2$.
IQR=Q3-Q1.
Retain data points within the range $[Q1 - 1.5 \times IQR, Q3 + 1.5 \times IQR]$.
```

4.2 Weight calculation

In this algorithm, n pose transformations, denoted as $T_{12}^{i}, \in [1, n]$, are obtained between points p_1 and p_2. Some of these pose transformations T_{12}^{i} are directly measured, while others are derived by multiplying multiple pose transformation matrices. As a result, different pose transformations T_{12}^{i} have varying levels of reliability.

We define the graph $G = [V, E]$, where the weight of each edge e_i is denoted as wiew_w_i^e. This weight represents the prior probability that the pose transformation T_{ei} corresponding to the edge eie_iei is an inlier, or it reflects the reliability of the pose transformation T_{ei}. The weight w_i^e can be pre-assigned based on the reliability or variance of the measuring sensor. Assuming that points p_1 and p_2 are connected via point p_3, the pose transformation between p_1 and p_2 can be expressed as follows.

$$T_{12} = T_{13}T_{32}$$

Let the weight corresponding to T_{13} be denoted as w_{13} (the probability of being a non-outlier), and the weight corresponding to T_{32} be denoted as w_{32}. Then, the weight corresponding to T_{12}, denoted as w_{12}, is calculated as follows:

$$w_{12} = w_{13}w_{32}$$

Under normal circumstances, it is required that T_{13} is a non-outlier and T_{32} is also a non-outlier; only then can the calculated T_{12} be considered a non-outlier. Consequently, the probability that T_{12} belongs to non-outliers is the product of w_{13} and w_{32}. However, there also exists a probability, albeit small, where T_{13} is an outlier and T_{32} is also an outlier, yet T_{12}, due to coincidence, turns out to be a non-outlier. For the sake of simplifying computation, we disregard the probability of this scenario. Here, we discuss the first case: when the pose transformation T_{12}^{i} corresponds to m edges $e_j, j \in S$, and each edge e_j is associated with a weight w_j^e; the weight w_{12}^{i} corresponding to the pose transformation T_{12}^{i} is given as follows:

$$w_{12}^{i} = \prod_{j \in S} w_j^e$$

Using the aforementioned steps, we can calculate the weights, w_{12}^{i}, corresponding to the n pose transformations, T_{12}^{i}, where $i \in [1, n]$.

4.3 Outlier removal in translation vector

We separate the rotation matrix R_{12}^{i} and the translation vector t_{12}^{i} from the pose transformation T_{12}^{i} and remove outliers sequentially. Below, we first describe the method for removing outliers from the translation vector $t_{12}^{i}, i \in [1, n]$. If the entire system operates on a plane, the translation vector $t_{12}^{i}, i \in [1, n]$ is as follows.

$$t_{12}^{i} = \left[tx_{12}^{i}, ty_{12}^{i}\right]^T \in R^2, i \in [1, n]$$

For each component of $t_{12}^{i}, i \in [1, n]$ (i.e., tx_{12}^{i} and ty_{12}^{i}), the weighted IQR (Interquartile Range) algorithm is applied sequentially to eliminate outliers. If any single component of a data point t_{12}^{i} is identified as an outlier, the entire data point t_{12}^{i} is considered an outlier and removed.

If the entire system operates in three-dimensional space, the translation vector $t_{12}^{i}, i \in [1, n]$ is expressed as follows.

$$t_{12}^{i} = \left[tx_{12}^{i}, ty_{12}^{i}, tz_{12}^{i}\right]^T \in R^3, i \in [1, n]$$

For each component of t_{12}^{i}, where $i \in [1, n]$ (i.e., tx_{12}^{i}, ty_{12}^{i}, and tz_{12}^{i}), the weighted IQR (Interquartile Range) algorithm is sequentially applied to remove outliers. If any single component is identified as an outlier, the entire data point t_{12}^{i} is considered an outlier and removed. The pseudocode for outlier removal of the translation vectors in 3D space is as follows.

Outlier removal algorithm in translation vector

```
Input: A set of translation vectors $t_{12}^{i} = \left[tx_{12}^{i}, ty_{12}^{i}, tz_{12}^{i}\right]^T \in R^3, i \in [1, n]$, a set of weights $w_i \in [1, n]$

Output: A filtered set of translation vectors $t_{12}^{i} \in S$ with
```

outliers removed

Apply the weighted IQR algorithm to the component tx_{12}^{i} to remove outliers, resulting in $tx_{12}^{i}, i \in S_1$.

Apply the weighted IQR algorithm to the component ty_{12}^{i} to remove outliers, resulting in $ty_{12}^{i}, i \in S_2$.

Apply the weighted IQR algorithm to the component tz_{12}^{i} to remove outliers, resulting in $tz_{12}^{i}, i \in S_3$.

Compute the final filtered set of translation vectors $t_{12}^{i}, i \in S, S = S_1 \cap S_2 \cap S_3$.

4.4 Outlier removal in 2D rotation matrix

If the entire system operates on a plane, the pose transformation matrix T is as follows.

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \in R^{3\times3}$$

$$R = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \in R^{2\times2}$$

We can convert the rotation matrix R into a single rotation angle, θ.

$$\theta = \arcsin(r_{12}) \in [0,2\pi]$$

In this approach, a set of n pose transformation matrices, denoted as $T_{12}^{i}, i \in [1,n]$, yields a corresponding set of rotation angles $\theta_i \in [0,2\pi], i \in [1,n]$. However, due to the periodic nature of rotation angles, directly applying an outlier removal algorithm to the set θ_i can lead to the following issue. if multiple θ_i values are distributed near 0, some angles may have values slightly greater than 0, while others may have values slightly less than 2π. Because the angle values cross the boundary, angles that are numerically very close can appear to have a large difference in value. To address this issue, we propose the following method: given a set of angles $\theta_i \in [0,2\pi], i \in [1,n]$, these angles are transformed into coordinates on the unit circle as $P = [\cos(\theta_i), \sin(\theta_i)], i \in [1,n]$. Since we assume that these data points are clustered around a single center, the transformed coordinates on the unit circle in the two-dimensional plane should also exhibit a single cluster. The two components in set P ($\cos(\theta_i)$ and $\sin(\theta_i)$) are processed sequentially using the weighted IQR algorithm to remove outliers. If any component is identified as an outlier, the corresponding θ_i is regarded as an outlier and removed. The pseudocode for this algorithm is as follows.

Outlier removal algorithm in 2D rotation matrix

Input: A set of rotation matrices $R_{12}^{i}, i \in [1,n]$, a set of weights $w_{12}^{i}, i \in [1,n]$

Output: A filtered set of rotation matrices $R_{12}^{i}, i \in S$

Convert the rotation matrix set into an angle set, $\theta_i \in [0,2\pi], i \in [1,n], \theta_i = \arcsin(r_{12}^{i})$;

Convert angles into unit circle coordinates, $[\cos(\theta_i), \sin(\theta_i)], i \in [1,n]$;

Apply the weighted IQR algorithm to the $\cos(\theta_i)$ component to remove outliers, resulting in a filtered subset $\cos(\theta_i), i \in S_1$.

Apply the weighted IQR algorithm to the $\sin(\theta_i)$ component to remove outliers, resulting in a filtered subset $\sin(\theta_i), i \in S_2$.

The final filtered set of rotation matrices is $R_{12}^{i}, i \in S, S = S_1 \cap S_2$.

4.5 Outlier removal in 3D rotation matrix

If the entire system operates in three-dimensional space, the pose transformation matrix T is as follows.

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \in R^{4\times4}$$

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \in R^{3\times3}$$

We can convert the rotation matrix R into a rotation vector v.

$$v = \theta n \in R^3$$

Where $n \in R^3$ represent a unit vector denoting the axis of rotation, and let $\theta \in [0,2\pi]$ represent the rotation angle. Following this representation, a set of n relative pose transformation matrices T_{12}^{i}, where $i \in [1,n]$, can be expressed as the following set of rotation vectors: $v_i = \theta_i n_i \in R^3$, where $i \in [1,n]$ and $\theta_i \in [0,2\pi]$. However, due to the periodic nature of rotation angles, directly applying an outlier removal algorithm to the set v_i, where $i \in [1,n]$, may lead to the following issue: if the values of θ_i for multiple v_i are distributed near 0, some angles may take values slightly greater than 0, while others may take values slightly less than 2π. Because these angles cross the boundary of the range, angles that are inherently close in value may exhibit large numerical differences. Consequently, the corresponding rotation vectors v_i in 3D space will appear to differ significantly in position.

To address this problem, we propose the following approach: first, identify the range of concentration for the rotation angles θ_i, where $i \in [1,n]$, in the set of rotation vectors v_i, where $i \in [1,n]$. Assume that the majority of the angle values θ_i, where $i \in [1,n]$, are distributed around θ_{ave}. Then, we can adjust the representation range of θ_i from $[0,2\pi]$ to $[\theta_{ave} - \pi, \theta_{ave} + \pi]$. After this adjustment, we apply a weighted IQR (Interquartile Range) algorithm to each component of the adjusted set of rotation vectors v_i, where $i \in [1,n]$, to remove outliers.

The specific processing steps are as follows: Given a set of angles denoted as $\theta_i \in [0,2\pi]$, where $i \in [1,n]$, these angles are transformed into coordinates on the unit circle: $P = [\cos(\theta_i), \sin(\theta_i)], i \in [1,n]$. Since we assume that the data has only one clustered center, the coordinates on the unit circle should also exhibit a single cluster center in the two-dimensional plane. Next, we apply the weighted IQR (Interquartile Range) algorithm to remove outliers from the two components of the set P (i.e., $\cos(\theta_i)$ and $\sin(\theta_i)$) sequentially. If any component is identified as an outlier, the corresponding θ_n is considered an outlier and removed. Let the remaining set of angles be represented as

$[\cos(\theta_i), \sin(\theta_i)], i \in S$. The mean position of the angular distribution can then be computed as follows.

$$\theta_{ave} = \frac{1}{|S|}\sum_{i\in S}\theta_i$$

Each element in the angle set is adjusted to the following form, $\theta_i \in [\theta_{ave} - \pi, \theta_{ave} + \pi], i \in [1, n]$. Subsequently, the rotation vector is adjusted as follows.

$$v_i = \theta_i n_i \in R^3, i \in [1, n]$$

$$\theta_i \in [\theta_{ave} - \pi, \theta_{ave} + \pi], i \in [1, n]$$

For each component of v_i, where $i \in [1, n]$ (i.e., v_i^1, v_i^2, and v_i^3), the weighted IQR (Interquartile Range) algorithm is sequentially applied to remove outliers. If any single component is identified as an outlier, the entire data point R_{12}^i is considered an outlier and removed. The pseudocode for removing outliers in the context of rotation matrices in 3D space is as follows.

Outlier removal algorithm in 3D rotation matrix

```
Input: Rotation matrix set $R_{12}^i, i \in [1, n]$, weight set $w_{12}^i, i \in [1, n]$
Output: Outlier-removed rotation matrix set $R_{12}^i, i \in S$
Convert the rotation matrices into rotation vectors, $v_i = \theta_i n_i \in R^3, \theta_i \in [0,2\pi]$.
Extract the set of rotation angles, $\theta_i \in [0,2\pi], i \in [1, n]$.
Convert the rotation angles to unit circle coordinates, $[\cos(\theta_i), \sin(\theta_i)], i \in [1, n]$;
Apply the weighted IQR algorithm to the cosine components, $\cos(\theta_i), i \in S_1$.
Apply the weighted IQR algorithm to the sine components, $\sin(\theta_i), i \in S_2$.
Compute the mean of the filtered angle distribution, $\theta_{ave} = \frac{1}{|S|}\sum_{i\in S}\theta_i, i \in S_1 \cap S_2$.
Transform the angle set to the range, $\theta_i \in [\theta_{ave} - \pi, \theta_{ave} + \pi], i \in [1, n]$.
Convert the rotation vectors back to the form, $v_i = [v_i^1, v_i^2, v_i^3]^T = \theta_i n_i \in R^3, \theta_i \in [\theta_{ave} - \pi, \theta_{ave} + \pi], i \in [1, n]$.
Apply the weighted IQR algorithm to the first component of the rotation vectors, $v_i^1, i \in Q_1$;
Apply the weighted IQR algorithm to the second component of the rotation vectors, $v_i^2, i \in Q_2$;
Apply the weighted IQR algorithm to the third component of the rotation vectors, $v_i^3, i \in Q_3$;
The outlier-removed rotation matrix set is $R_{12}^i, i \in Q_1 \cap Q_2 \cap Q_3$.
```

5. Path planning algorithm

5.1 Dijkstra algorithm

The algorithm requires finding multiple pose transformations, denoted as T_{12}^i, between points p_1 and p_2, where $i \in [1, n]$. This problem can be regarded as searching for multiple paths from p_1 to p_2, which can be solved using the Dijkstra algorithm. The Dijkstra algorithm does not utilize heuristic functions to guide its search when determining paths. Starting from point p_1, the algorithm expands outward, calculating the shortest path cost to each neighboring point, and terminates once it reaches p_2. However, in this algorithm, we typically restrict the connection between p_1 and p_2 to pass through at most one or two intermediate points. If the path between p_1 and p_2 involves too many intermediate points, the accumulated pose transformation between p_1 and p_2 may become inaccurate due to noise. Therefore, even though the Dijkstra algorithm lacks heuristic guidance, the additional computational cost remains minimal when the search is limited to short paths.

We define the weight of each edge e_i in the graph $G = [V, E]$ as w_i^e. This weight represents the probability that the pose transformation T_{ei} associated with the edge is an inlier, or alternatively, it indicates the reliability of the pose transformation T_{ei}. The weight w_i^e can be pre-assigned based on the reliability or variance of the measurements obtained from the sensors.

Assuming points p_1 and p_2 are connected through point p_3, the pose transformation between p_1 and p_2 is expressed as follows.

$$T_{12} = T_{13}T_{32}$$

Let the weight corresponding to T_{13} be w_{13} (the probability of being a non-outlier), and the weight corresponding to T_{32} be w_{32}. Then, the probability w_{12} of T_{12} being a non-outlier is given as follows.

$$w_{12} = w_{13}w_{32}$$

We need to find a path from p_1 to p_2 that maximizes the cumulative weight (the probability of being a non-outlier). However, the error in the above form is difficult to apply directly to path-planning algorithms, such as the Dijkstra algorithm, which requires the error along the path to be in an additive form. Therefore, we take the logarithm of the above error and multiply it by -1, resulting in the following.

$$-\log(w_{12}) = -\log(w_{13}w_{32})$$

$$-\log(w_{12}) = -\log(w_{13}) - \log(w_{32})$$

We use the negative logarithm of the weight of each edge, denoted as $-\log(w_i^e)$, as the edge weight in the path planning algorithm. This transformation ensures that the edge weight is additive, which aligns with the cumulative nature of path cost calculations. Before applying the negative sign, the algorithm seeks to maximize the path cost w_{12}. After applying the negative sign, it seeks to minimize $-\log(w_{12})$, which satisfies the requirement of finding the shortest path in path planning algorithms. Additionally, since the original edge weight w_i^e is within the range $[0,1]$, its transformed value $-\log(w_i^e)$ lies within the range $[0, +\infty]$, which ensures compliance with Dijkstra's algorithm's requirement that edge weights be non-negative.

In this algorithm, we aim to find n paths from p_1 to p_2 such that these paths are as short as possible (i.e., have minimal path costs) while minimizing the use of overlapping edges. The reason for avoiding overlapping edges is that if two paths share a significant number of edges, the independence of the pose transformations between p_1 and p_2 corresponding to these paths will be low. This lack of independence means that these paths cannot adequately represent two independent measurements of the pose transformation between p_1 and p_2.

To address this, we can adopt the following method to obtain n paths. First, use the Dijkstra algorithm to find the shortest path from p_1 to p_2 in the graph $G = [V, E]$. Then, for all edges on this path, set their weights to a very large value Δ (e.g., 10^5), while keeping the weights of all other edges unchanged. Next, use the Dijkstra algorithm again to find the shortest path from p_1 to p_2. At this stage, the path planning algorithm will attempt to avoid using the edges that were part of the previous path (as their costs are now Δ) unless it is absolutely necessary to use certain edges to connect p_1 to p_2. This process is repeated iteratively. Each time a new path from p_1 to p_2 is found, the weights of the edges in the newly discovered path are updated to Δ. The algorithm terminates when either n paths are found, or when a new path is found where all edges are repeated edges (i.e., the path cost equals $k \times \Delta$, where k is the number of edges in the path). The pseudocode for this algorithm is as follows.

Using Dijkstra's algorithm to find n paths

```
Input: Graph $G = [V, E]$, edge weights $w_i^e, i \in [1, |E|]$, the desired number of paths n
Output: k paths from $p_1$ to $p_2$, $k \leq n$
Compute the negative logarithmic weights: $\omega_i^e = -\log(w_i^e), i \in [1, |E|]$.
For (j in 1 to n):
    Use $\omega_i^e$ as edge weights and apply the Dijkstra algorithm to find the shortest path from $p_1$ to $p_2$. Denote the path cost as C[j] and the path as Path[j].
    Set the weights of all edges in Path[j] to $\Delta$ ($10^5$).
    If(number of edges in Path[j] $\times$ $\Delta$=C[j]):
        Remove Path[j], break.
Output k paths from $p_1$ to $p_2$, $k \leq n$.
```

4.2 A* alorithm

The algorithm requires finding multiple pose transformations, $T_{12}^i, \in [1, n]$, between points p_1 and p_2. This problem is equivalent to searching for multiple paths from p_1 to p_2. In addition to using the Dijkstra algorithm, the A* algorithm can also be employed. The A* algorithm evaluates each node on the path using the following objective function:

$$f(n) = g(n) + h(n)$$

The function $f(n)$ represents the total estimated cost from the starting point p_1 to the endpoint p_2. The function $g(n)$ denotes the actual cost from the starting point p_1 to node n (i.e., the sum of the weights of all edges traversed so far). The function $h(n)$ is a heuristic function used to estimate the cost from node n to the endpoint p_2. This heuristic function $h(n)$ provides directional guidance toward the endpoint, enabling the algorithm to reach p_2 more efficiently. Without the use of $h(n)$, the A* algorithm would degrade into Dijkstra's algorithm.

For the heuristic function $h(n)$ in the A* algorithm, the Euclidean distance can be used as its form.

$$h(n) = \lambda\sqrt{\left(x_n - x_{p2}\right)^2 + \left(y_n - y_{p2}\right)^2}$$

The heuristic function requires prior knowledge of the 3D coordinates of each node in the graph $G = [V, E]$. These coordinates can be roughly estimated by first computing a preliminary set of node coordinates using the graph $G = [V, E]$ without removing outliers. For the A* algorithm, the acceptability of the heuristic function requires that $h^*(n)$ must not exceed the true cost $h^*(n)$; otherwise, the path found may be suboptimal. Since the node coordinates are obtained from the graph $G = [V, E]$ without removing outliers, directly using the Euclidean distance as the heuristic function may result in $h(n)$ being greater than the true cost $h^*(n)$. To address this, we multiply the Euclidean distance by a factor $\lambda \in [0,1]$. When applying the A algorithm to search for the path from p_1 to p_2, the other steps are identical to those used in the Dijkstra algorithm.

REFERENCES

[1] Agarwal, P., Tipaldi, G. D., Spinello, L., Stachniss, C., & Burgard, W. Robust Map Optimization using Dynamic Covariance Scaling. IEEE International Conference on Robotics and Automation (ICRA), 2013.

[2] Aravkin, A. Y., Burke, J. V., Chiuso, A., & Pillonetto, G. Sparse and Robust Estimation: A Nonconvex Optimization Approach. Automatica, 2017.