**同步欧氏距离（Synchronized Euclidean Distance,SED）**[6] 同步欧氏距离是指原始轨迹点到对应压缩轨迹段的时间比例距离，与垂直欧氏距离不同的是，同步欧氏距离考虑时间参数。跟上图类似，假设原始轨迹序列中（xi, yi,ti,fi）到（xj,yj,tj,fj）是一条轨段，i<j，该轨迹段时间序列内的任意一点（xk, yk,tk,fk）到该轨迹段的投影位置为（）。则（xk, yk,tk,fk）在轨迹段上按照时间比例的投影位置为（），二者之间的距离称为同步欧氏距离SED，如图2-2所示。

（,,）

（,,）

（，）

（,,）

SED

**图2-2** 同步欧氏距离说明图

投影位置（）的位置计算如下：

**(2-2)**

**(2-3)**

算法原理

在数据压缩阶段，首先需要用户提供精度要求（压缩阈值）和轨迹数据的采集周期。将采集的第一个点直接保存为特征点，并且将其设置为锚点作为窗口的起点。从第二个点开始，计算锚点到该点之间所有点到有锚点和该点所连线的同步欧氏距离（SED），若SED小于阈值则窗口继续先后移动，若SED大于阈值则将该点前一个点保存为特征点并设置为新窗口的锚点而后从新的窗口开始移动，直到整条轨迹都背遍历，在算法最后保证最后一个轨迹点被保存为特征点，则完成整个压缩过程，压缩后的轨迹将会由一系列的特征点构成。

如图3-1所示，p1、p2、p3、p4、p5、p6、p7、p8位原始轨迹点，此时p1是窗口起始锚点也是第一个特征点，p7为此时窗口的终点，、、、、为窗口内对应点在此窗口情况下对应的相对模拟恢复点的位置。模拟恢复点通过计算窗口终点和起点之间的目标的移动速度和时间差来预估丢弃点的个数并通过速度和预设采集周期T模拟在直线方向上生成模拟恢复点。此时窗口内各个点到其对应模拟生成点的距离，即为同步欧氏距离di，若窗口内的所有di均小于预设距离阈值D则窗口继续向下一个点移动，如图3-1所示，其窗口内所有点的di均小于预设距离阈值D，则窗口继续移动即以p8作为窗口终点进行判断，就生成了如图3-2的轨迹状态。只要窗口内有一个点的di大于预设距离阈值D，则判断轨迹有较大的偏移，将此点的上一个点保存为特征点并设为新窗口的初始锚点，重置窗口大小开始新一轮的窗口判断，如图3-2所示，此时p6和p7的同步欧氏距离大于阈值，压缩算法则会将p7保存为特征点并设置为窗口新的锚点并且重置窗口大小，由p8开始一轮新的滑动窗口判别。

由两个示例也可以知道，在窗口发生变化之后窗口内各个点的模拟生成的位置均会发生变化故而每次窗口移动均要伴随着窗口内所有点的模拟恢复点的重新预估和同步欧氏距离的重新计算会造成一点的时间损耗。由示例也可以看出在实现图中p1到p7的轨迹压缩的时候中间p2到p6将会被丢弃只需要保留p1和p7两个点即可以近似的描述这个轨迹段在一定程度上保证了数据的压缩率。

图3-1 压缩轨迹示例1

×

×

图3-2 压缩轨迹示例2

在数据恢复阶段，系统将会优先读取第一、二个特征点，并且将第一个特征点直接恢复，之后开始这两个特征点之间丢弃的点的数量，若存在丢弃点，依次恢复出窗口内所有非特征点，完成非特征点恢复后恢复第二个特征点并后读入下一个特征点开始新一轮判断。该过程循环进行，直至恢复出最后一个特征点。

在恢复阶段可能存在恢复点多于或者少于原始点的情况。恢复点多于原始点的情况会发生在轨迹采集周期小于两个原始点之间的时间间隔，针对此情况本系统允许一定时间范围内的预测来尽可能实现在原始点有丢失的情况下的轨迹还原。只要在三个时间周期的内的时间间隔这按正常情况进行压缩和恢复，若大于三个时间周期则将这两个点均保存为特征点，并且在第二个点设置标识，标识这个特征点与前一个点之间没有压缩点，来保证恢复时不误恢复。恢复点少于原始点的情况则正相反会发生在轨迹采集周期大于两个原始点之间的时间间隔，这种情况发生的概率比较低，少数的点可以忽视，在若出现大量的情况则需要考虑两个方向，一是改变压缩周期，二是通过模拟点和原始点的同步欧氏距离来自适应的完成轨迹压缩，虽然还是会有一定的轨迹点丢失但是可以保证轨迹轮廓的基本不变。

压缩阶段

在轨迹压缩阶段我们主要需要注意生成模拟恢复点和计算同步欧氏距离两个主要的部分，下面对这两部分的公式进行介绍。

**(3-1)**

在求模拟恢复点的时候需要求得窗口的大小N，通过式(3-1)即可求得，pj和pi为窗口的两个端点，T为默认的采集/存储周期。

**(3-2)**

**(3-3)**

在模拟恢复轨迹点中我们首先要确定窗口的起点pi和终点pj,并且需要窗口大小N（即窗口中点的个数，包含窗口终点不包含窗口起点）和当前点的窗口位置n。有了上述的条件之后，可以通过三角形相似的性质计算窗口中模拟生成的轨迹点的位置，如式(3-2)和(3-3)所示，通过三角形相似对线段进行等比例缩放分别求出模拟恢复点的x，y位置，在附带保存轨迹的额外信息即可生成恢复点。

**(3-4)**

在生成恢复轨迹点之后需要求其同步欧氏距离以用于判断窗口终点是否为特征点，此时pk为原始轨迹点，为恢复点的位置，通过式(3-1)可以求得两点之间的距离，该距离即为同步欧氏距离。

在压缩判断中我们首先使用式(3-2)(3-3)来生成恢复轨迹点，而后使用式(3-4)来生成pk和的同步欧氏距离，并将其与预定距离阈值D进行比较判别来确定窗口内的点是否满足特征点的要求。

详细压缩算法如表3.1所示

**表3-1** 压缩算法

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 压缩算法：基于Open Window和同步欧式距离的压缩算法 | | | | | |
| 输入：原始轨迹：traces（p1,p2,…pn）压缩阈值D 轨迹存储周期T | | | | | |
| 输出：压缩后的轨迹集：Tra | | | | | |
| Compression（traces，D，T）开始 | | | | | |
|  | //p1保存并设置为窗口起始点pt，窗口位置n | | | | |
| p1→Tra；p1→pt；1→n； | | | | |
| For i=1 to n do | | | | |
|  | //判断轨迹点的楼层是否发生变化，轨迹点的时间差是否远大于周期T | | | |
| If pt.floor!=pi.floor OR timei-timei-1>>T then | | | |
|  | //保存pi-1和pi，并将pi设置为窗口起始点 | | |
| pi-1→Tra；pi→Tra；pi→pt；i→n； | | |
| Else then | | | |
|  | For j→n+1 to i-1 do | | |
|  | //判断窗口内的点的同步欧式距离是否大于阈值 | |
| SED→getSED（pn，pj，pi） | |
| If（SED>D） | |
|  | pi-1→Tra；i-1→n； |
| If（n=i-1） | |
|  | pi→Tra； |
| Return Tra | | | | |

恢复算法

恢复算法从接受的第二个特征点开始就需要进行判断，首先判断点之间是否有丢弃的点，若有丢弃点则在通过式(3-1)计算上一个特征点和本点之间的窗口大小N，而后循环调用式(3-2)和(3-3)来生成窗口中的恢复点，在恢复完成后在恢复本特征点读入下一个特征点，直到恢复最后一个特征点。在循环过程中要额外注意，在计算窗口大小时两个相邻的原始定位点之间也有周期1，但其中不存在模拟恢复点，所以要求我们在循环是循环结束条件需要额外减一。

详细恢复算法如表3-2所示

**表3-2** 恢复算法

|  |  |  |  |
| --- | --- | --- | --- |
| 恢复算法：基于Open Window和预测点的恢复算法 | | | |
| 输入：压缩后的轨迹：Tra（p1,p2,…pn）轨迹存储周期T | | | |
| 输出：恢复后的轨迹集traces | | | |
| Restore（Tra， T）开始 | | | |
|  | //p1保存 | | |
| p1→traces； | | |
| For i=2 to n do | | |
|  | //根据存储周期判断窗口间原有点个数n | |
| n→getNum（pi-1，pi） | |
| //窗口内循环恢复点 | |
| for j→1 to n-1 do | |
|  | //生成模拟点 |
| pt→getSimulatpoint（pi，pi-1，j）； |
| pt→traces； |
| pi→traces； | |
| Return traces | | |