**1.Introduction**

轨迹跟踪(Trajectory Tracking)由[]提出，是位置跟踪(Position Tracking)技术和轨迹压缩(Trajectory Compression)技术的结合。位置跟踪和轨迹压缩原本是两个独立的任务，分别实现各自的目标。其中位置跟踪源于航海……，旨在以较小的代价确定移动对象的当前位置，本质上是移动对象（MO）和MOD之间达成的一种协议，通常是双方基于移动对象（Moving object）的历史信息（位置与速度等）来推测其当前位置，从而减少MO和MOD之间的信息交换量。随着智能手机、智能手表、车载系统的普及以及定位技术（如GPS等）在移动设备中的广泛应用，使得我们可以更加便捷的跟踪移动物体（Moving object）（卫星、船舶、车辆、行人等）的位置。

~~MOD对MO的轨迹跟踪则需要MO持续地将其当前位置点信息通过无线网络传输给MOD。一种比较极端的做法就是MO不断的将其GPS设备采集到的所有位置点传输给MOD，MOD通过其轨迹信息对其历史位置进行查询，并通过最近得到的位置点求出其速度从而对当前位置点进行预测。~~

另一方面，轨迹压缩技术则是通过去除物体（Moving object）历史轨迹中的冗余点，减少了位置信息的传输和存储开销。轨迹轨迹是MO历史位置按时间排序的序列。将MO的轨迹数据存储下来可以进行很多研究，例如可以对object的历史位置进行查询、分析等，因此车辆信息服务、位置服务上都需要保存MO的轨迹信息。但是随着现如今GPS采样频率的增加，传输大量的采样点会对网络带宽和MOD的存储空间造成很大的压力。如果简单粗暴地直接降低采样频率来减少传输的轨迹点的数量，那么我们所保留的轨迹会与物体真实运动轨迹之间将存在较大的偏差。轨迹压缩技术则是为此而提出。

如上所述，Position Tracking旨在以较小的信息交换达到追踪MO的目的，Trajectory旨在减小轨迹的规模,它们原本是独立的两项技术。但是学者[]发现，如果把两项技术相结合，则既可以追踪移动对象的位置，又可以减少信息传输和存在的代价。

在论文[1]中，作者通过修改Line Dead-Reckoning~~(一种最早用于轮船航迹追踪的方法)提出了LDR方法，实现了对MO不超过给定误差范围的位置追踪，但是其压缩后的轨迹与原始轨迹可能会超出。随后，作者证明LDR压缩后轨迹与原始轨迹误差不会超过2，于是将其位置跟踪阈值设为，得到了迹与原始轨迹误差不超过的压缩轨迹，这就是改进后的LDRH算法~~。提出了LDRH算法，并证明该算法可以同时跟踪MO的位置和压缩轨迹数据。~~但是LDRH在跟踪过程过于严格，导致轨迹压缩效果不好~~。LDRH是个非常轻量的协议，具有O(1)的空间复杂度和O(n)的时间复杂度。其不足之处是压缩效果（压缩率）差，意味着MO和MOD任然需要传输和保存较多的数据。针对LDRH的这个问题，论文[2]和[3]提出了CDR（Connection-Preserving Dead Reckoning）算法，它通过增加缓存以及更加细致的信息更新技术，一定程度上提高了压缩率。不过CDR的压缩算法依然依赖于LDR，存在压缩率低的问题。对此，作者进一步提出了GRTS（Generic Remote Trajectory Simplification）框架，它把轨迹压缩和位置跟踪分解为两个子过程：(1)使用Dead Reckoning来进行位置跟踪，(2)使用已有的算法来实现轨迹压缩。相比LDRH和CDR，GRTS框架通过集成压缩效果更好的轨迹压缩算法，减少了信息传输和存储的数量。不过GTRS为此引入了缓存，且其集成的轨迹压缩算法具有较高的空间复杂度和时间复杂度，这都不利于其在资源受限的移动设备(手机、智能手表等)上运行。

**~~Problem analysis~~**~~：MOD对MO的轨迹跟踪分为轨迹压缩和位置跟踪两部分。位置跟踪是指根据物体（Moving object）的历史位置速度等信息来对其当前位置进行预测。轨迹压缩则是去除物体（Moving object）历史轨迹中的冗余点，将其关键点传输至Moving objects databases(MOD)中，保留其轨迹信息，用作历史位置查询等用途。不同的轨迹跟踪方法都是在位置追踪准确性，轨迹压缩效果，信息传输量，计算开销，MO缓存空间等之间做出权衡（trade off）。~~

**Contribution**:在本文中，我们受轨迹压缩算法CISED的启发，设计出了一种新的轨迹跟踪算法one pass linear tracking（OPLT），它既有与LDRH一样的低时间和空间复杂度，又有与GRTS类似的压缩率和数据传输量。具体而言，本文的主要贡献如下：

1. 我们设计了一种新的轨迹跟踪算法one pass linear tracking（OPLT），该算法将位置追踪和轨迹压缩结合为一个统一的过程，具有在O（n）时间复杂度和O（1）空间复杂度，并且具有良好的压缩效果，在MO和MOD之间传输很少的信息。

2.我们改进了轨迹压缩CISED……更好的压缩率；

3.我们将（算法名字）与LDRH，CDR，GRTS等算法在四个（Geolife，Mopsi，Taxi，Ucar）真实轨迹数据集上进行了实验，从运行时间，MO与MOD之间传输信息量，轨迹压缩效果等几个方面进行了对比。

~~3.我们实现了轨迹追踪系统，我们的算法运行在移动设备（android手机）上，将手机GPS设备采集到的轨迹点进行判断，将必要的信息（轨迹点、速度）传输到MOD中，MOD根据得到的信息保留压缩后的轨迹并对MO当前位置进行预测。~~

**2.preliminary**

在这部分，我们介绍一些关于轨迹追踪的基础知识。轨迹跟踪主要包括位置跟踪和轨迹压缩两个过程。我们将首先分别介绍位置跟踪和轨迹压缩的定义，然后介绍一些当前主要的轨迹跟踪算法。

**2.1 Position Tracking: Linear Dead-Reckoning**

(首先用一段话概括Position Tracking,主要工作；其次重点介绍LDR的原理和特点)

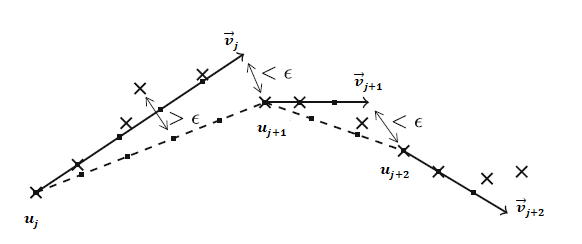


图1 LDR示意图

Dead-Reckoning 是一种策略（方法）通过对MO当前位置进行预测来减少MO与MOD之间传输的信息量。如图1所示，**X**表示MO中GPS设备采集到的实际轨迹点，初始时，MO将及其速度矢量传输至MOD中，当MO收到GPS设备采集到的轨迹点时，将实际轨迹点与根据及其速度求得的同步轨迹点进行比较，如果其距离不超过给定阈值，则与MOD间不进行信息的传递。当GPS接收到的实际轨迹点与其预测点距离超过，则将其上一个实际位置点，即最后一个未超过阈值的点（图中）及其速度矢量传输至MOD，进行下一次跟踪。MOD中接收到的轨迹点连接起来得到连续的折线段,即为压缩后的轨迹。这就是Line Dead-Reckoning(LDR )算法。（在实际运用中，我们一般根据连续的两个点位置变化求出其速度矢量）

**2.2 Trajectory simplification: CISED**

(首先用一段话概括**Trajectory simplification**,主要工作；其次重点介绍CISED的原理和特点)

**2.3 Trajectory tracking**

**（首先用一段话概括Trajectory tracking,其次介绍LDRH的原理，最后介绍CRTS的原理）**

轨迹跟踪主要分为位置跟踪和轨迹压缩两个部分，位置跟踪的目的是根据历史信息对MO当前位置进行预测，而轨迹压缩则是将原始轨迹中关键轨迹点保留下来。

**2.3.1 LDRH**

LDR实现了轨迹的位置跟踪，但是其保留的压缩轨迹不符合要求。LDRH算法虽然能够得到符合要求的压缩轨迹，但是其位置跟踪过程阈值的设定过于严格，使得压缩后的轨迹效果不好。MO与MOD中仍然需要传递大量信息。

**2.3.2 GRTS**

~~在论文[3]中，作者在LDR的基础上提出了CDR算法，为了解决LDR算法保留的压缩轨迹中存在与原始轨迹点超过阈值的问题。CDR算法中更加严格的限定了跟踪过程的更新条件。在MO中设置缓存，保存上次进行信息传输后GPS采集到的所有原始轨迹点，每次接收到新的GPS点时，将此时的轨迹点与缓存中起点连成线段，判断缓存中其他点到线段的同步距离是否超过阈值，如果超过，则进行一次更新。同LDR一样，如果位置跟踪过程超过阈值，也进行一次更新。但是在极端情况下可能导致缓存中存的点的数量一直增长，需要缓存无限大。为了解决这个问题，提出了改进的算法，将MO中的缓存空间设为固定大小。当存储的轨迹点满时，对缓存中的点进行优化，删去其中一些冗余点。~~

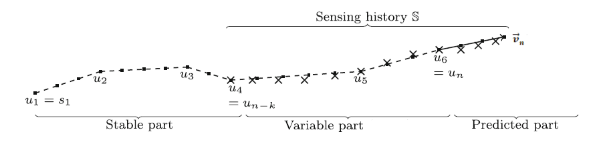


图2 GRTS框架示意图

~~LDRH和CDR算法虽然都能实现轨迹跟踪，但是都存在共同的局限性，即轨迹压缩的效果依赖于位置跟踪的准确性。因此作者提出了~~GRTS框架，分离了位置跟踪和轨迹压缩两个过程。如图2所示，GRTS将整个过程分为三个部分，Stable part，Variable part，Predicted part。其中，stable part是指已经存储在MOD中的已经压缩后的轨迹。Variable part是位于MO中的缓存空间，存储一些最近的真实轨迹点可以结合其他轨迹压缩方法对轨迹进行压缩，将压缩后的点传输至MOD中。Predicted part是位于MO和MOD中的采用Dead-Reckoning方式进行位置跟踪的部分。

**3.Problem Statement**

资源受限环境的要求该算法

问题的形式化描述？输入、输出,满足的条件

**4.Method**

首先我们对当前采用同步距离的轨迹压缩算法中空间复杂度和时间复杂度都很低的CISED-S算法进行了改进，并在此基础上增加位置跟踪过程实现了新的轨迹跟踪算法。在位置跟踪过程中对预测速度进行调节，尽可能地减少了 MO传输至MOD中的信息量，在保证位置跟踪误差的前提下得到压缩效果良好的压缩轨迹。

**3.1 GRTS + CISED**

GRTS是一种实现traj tracking的框架，它~~将整个轨迹追踪过程分为三个部分，Stable part，Variable part，Predicted part。~~分离了追踪和压缩两个过程，在Predicted part进行位置跟踪，在Variable part结合轨迹压缩算法进行压缩。

CISED-S是一种采用同步距离的压缩效果良好的轨迹压缩算法【】。因此我们尝试将CISED与GRTS结合，我们将其运用在GRTS的Variable part。CISED-S是一种one pass算法，因此GRTS框架中Variable part再次压缩并不会修改压缩后的点。同时，我们发现CISED-S虽然压缩效果已经足够好，但是仍然存在着一些不足。

**3.1.1 CISED-S改进**

在此基础上，我们认为CISED-S算法在建立时空锥的过程中的半径取值过于严格。我们将阈值设为，如图3所示：当与存在交集，然而线段在时刻的同步点与真实轨迹点之间距离超过阈值。为了解决这个问题，我们更加严格限制了其更新条件，如果我们保证穿过斜圆锥，即可保证线段在时刻的同步点与真实轨迹点之间距离不超过阈值，如图4所示。

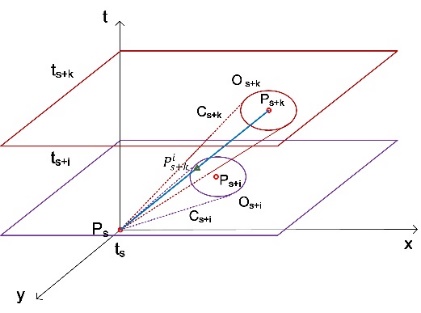
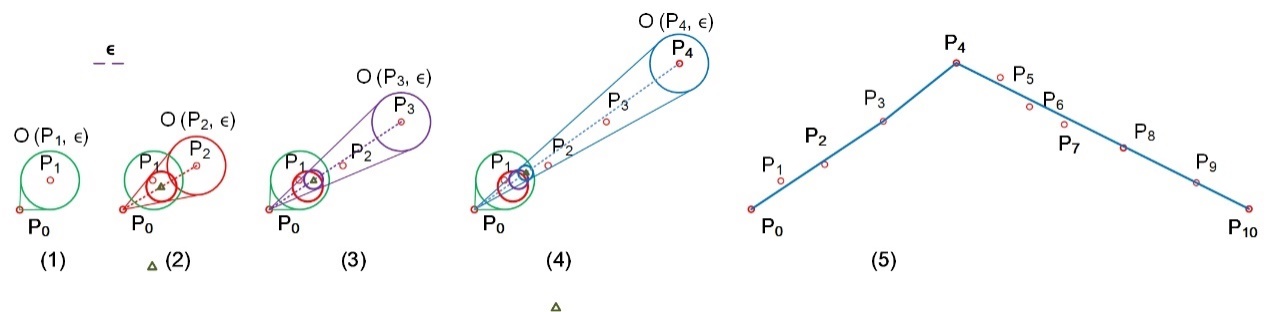


图4 时空锥示意图

因此，给定三个点，,。以为顶点，为底面建立斜圆锥。若穿过斜圆锥，即可保证线段与之间的同步距离不超过，如图5所示。 即给定三个点，,。若与在平面上的投影圆与存在交集S=。则与存在公共区域。

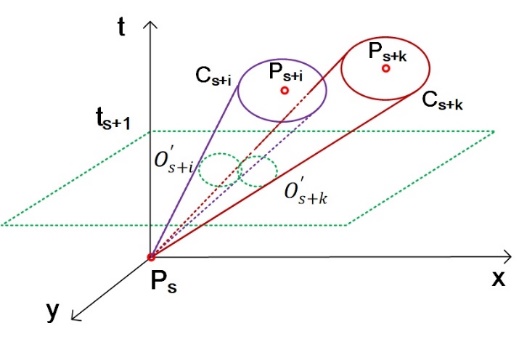


图5 时空锥示意图

**证明**：给定轨迹。以为顶点，为底面建立时空锥。若在平面存在交集S=。则存在公共交集。在时刻同步点为，若，则说明穿过S，则均穿过圆锥。由定理1可知，中所有点到线段的同步距离均不超过阈值。因此可作为的一段压缩轨迹。得到定理1。

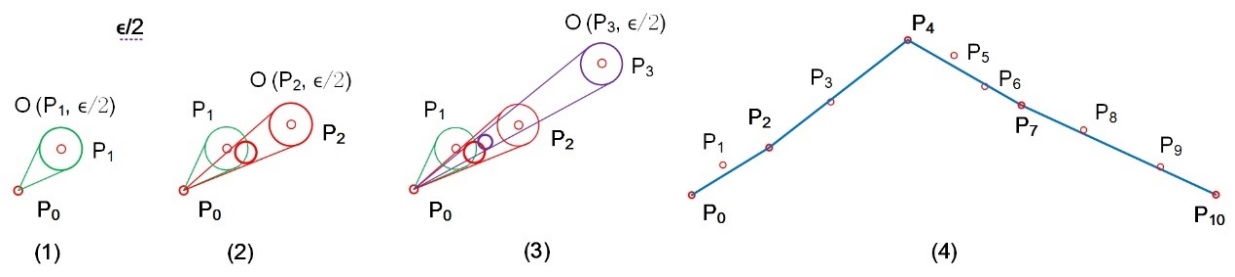
**定理1：**给定轨迹，以为坐标原点，若时空锥集合，在平面存在交集S=。线段在时刻同步点，则说明可作为的一条误差不超过的压缩轨迹。

图7 CISED-示意图

图6 CISED-S示意图

因此CISED-算法的执行过程为：初始时，以为顶点建立坐标系，当接收到点时，以为圆心，为半径做圆，令S=。每当接收到点时，则判断在时

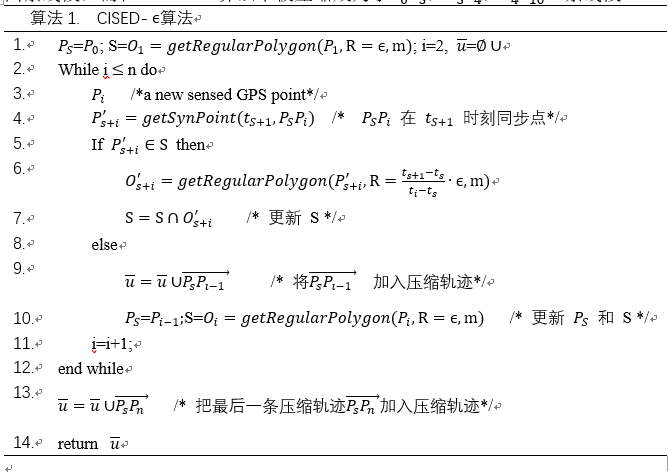
刻的同步点是否在S中，如果在，则找到的投影圆。由于在S中，则与S必有交集，更新。如果不在S中，则说明中存在点到线段的同步距离超过阈值。则更新为的一条压缩轨迹，同时以为原点建立坐标系继续压缩。

图6和7给出了CISED-S和CISED-在同一段轨迹上的示意图。在CISED-S算法中，当到来时，与和构成的公共区域没有交集，故进行一次压缩，将线段作为，，的压缩轨迹。而在CISED-算法中，由于斜圆锥底面圆半径更大，所以圆锥更容易存在交集，与和构成的公共区域存在交集，同时穿过交集部分，所以不会被压缩，当时，不符合条件，因此进行压缩。在CISED-S算法下被压缩成了，，，四条线段，而在CISED-算法下被压缩成为了，，三条线段。

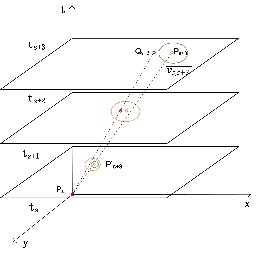
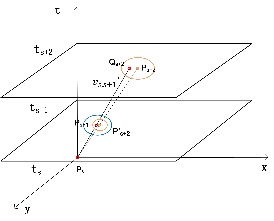
CISED-算法伪代码如图：

**3.2 one pass line tracking（OPLT）**

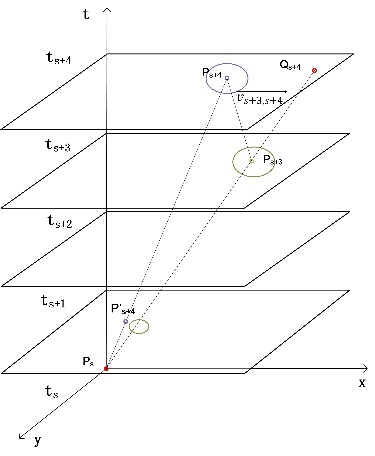
**3.2.1算法原理**

CISED-是一种one pass算法，因此GRTS框架中Variable part再次压缩是多余的。因此我们尝试在CISED-的基础上，增加对移动对象的位置跟踪功能，实现一种one pass line tracking（OPLT）。相较于LDRH算法，OPLT大大的减少了移动对象与服务器中的传输的信息量，从而得到了压缩效果更好的压缩轨迹。相较于算法，OPLT减少了轨迹跟踪过程中在移动对象中的缓存开销，同时减少了传输的信息量从而得到效果更好的压缩轨迹。相较于算法，OPLT大大减少了计算时间，并得到了与其压缩率基本一致的压缩轨迹。

我们将位置追踪与轨迹压缩两个过程结合起来，在位置跟踪的同时，实现轨迹压缩。位置跟踪最关键的就是增加对移动对象的速度预测，像传统轨迹跟踪算法一样，我们初始时以及其邻近的点求得其速度作为其原始速度。不同的是我们在位置跟踪过程中会对其速度进行调整。我们以历史轨迹点为坐标原点建立三维直角坐标系，通过坐标原点位置及速度对移动对象当前位置做出预测，如果预测点与实际GPS接收到的点误差不超过给定阈值，则继续追踪过程，如果超过给定阈值，则判断中是否有点到线段的同步距离超过。如果没有超过，则修改处的预测速度,继续追踪过程。如果中有点超过阈值，则将传输至MOD中，并以为坐标原点建立坐标系，=，继续进行轨迹跟踪。



1. （2）



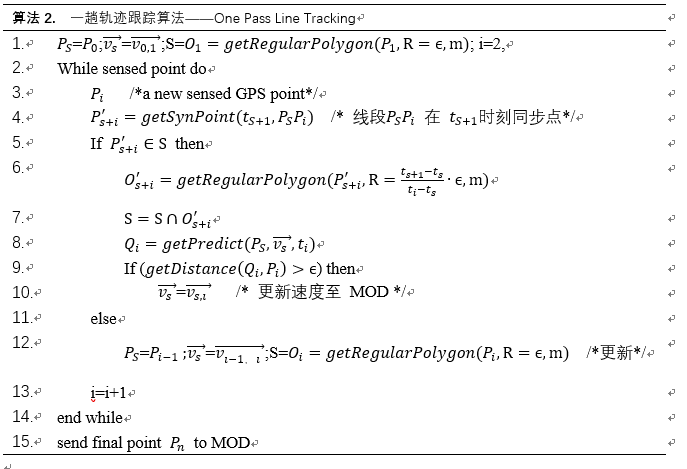
（3）

图8 一趟轨迹追踪算法执行过程图

如图8-(1)所示,初始时，以时刻的实际位置点为坐标原点，建立三维坐标系，当时刻时，在移动对象接收到GPS传感器采集到的实际位置点，然后根据和，求得速度传输至服务器中, 其中速度方向为，速度大小||=。以为圆心，给定阈值为半径得到圆，初始时令S=。

在时刻，根据点及速度，对时刻位置进行预测，坐标为。当移动对象采集到实际的GPS位置时，连接，求出此时时刻同步点坐标。在S中，则计算的同步圆。更新S=S∩。计算点与点之间的距离，小于给定阈值，则继续以原始速度进行跟踪。

如8-(2)所示，在时刻，根据点及速度，对时刻位置进行预测，坐标为。当移动对象采集到实际的GPS位置时，连接，求出此时时刻同步点坐标。在S中，则计算的同步圆。更新S=S∩。计算点与点之间的距离，超过给定阈值，则修改对点的速度预测，修改速度为, 其中速度方向为，速度大小||=。移动对象将速度传输给服务器。

如8-(3)所示,在时刻，根据点及速度，对时刻位置进行预测，坐标为。当移动对象采集到实际的GPS位置时，连接，求出此时时刻同步点坐标。不在S中，此时，移动对象将点及速度传输给服务器，以为坐标原点建立三维坐标系继续进行轨迹追踪。

在每次获得时，我们都在线段上找出时刻的同步点。如果位于S中，由定理3可知，可以作为的一条压缩轨迹，更新。如果超出区域S，则说明如果以作为的一条压缩轨迹，在中存在点到的同步距离超过，则将点及速度传输至服务器中，保留在服务器中作为的压缩轨迹。以为坐标原点，为速度接着进行轨迹跟踪。从而保证了压缩后的轨迹与原始轨迹误差不超过阈值。在位置跟踪过程中，会判断根据和得到的当前时刻同步点与当前实际位置点距离是否超过，如果没有超过，则说明在服务器中对当前移动对象位置点的预测与其实际位置偏差没有超过阈值，继续追踪过程。如果超过，则说明对当前移动对象位置点的预测与其实际位置偏差超过了阈值，故需要对其预测速度进行调整，调整=，传输至服务器中，继续追踪过程。从而保证了我们的追踪过程与压缩得到的轨迹误差均不超过阈值。

**3.2.2算法伪代码**

算法2为一趟轨迹跟踪算法的伪代码。当移动对象开始运动时，首先将传输至服务器中为，在时刻，将得到的速度=传输至移动对象中。在接下来的过程中，每当采集到新的位置点时，根据连线求得在时刻的同步点，如果不在区域S中，则说明当前进行预测的和均需要更新，更新==, S=。如果在区域S中，则根据和对当前位置点进行预测得到，比较与间距离，若超过阈值，则更新至服务器中。随后，更新。继续轨迹追踪过程。

由于我们的轨迹跟踪算法是在CISED-算法基础上演变过来，因此算法时间复杂度和空间复杂度都很低。同时，传统的轨迹追踪算法，每次移动对象进行更新时都是传输一个位置点及其速度至服务器中。而在我们的轨迹跟踪算法中，当位置跟踪超出阈值时，只要中所有点到线段的同步距离均不超过，则说明如果仍然以为原点，调整速度方向为，仍然可以进行追踪过程，无需传输点进行轨迹压缩。所以在整个轨迹追踪过程传输了更少的信息量到服务器中。我们最终在计算时间，传输信息量都较低的情况下，得到了压缩效果良好的压缩轨迹。

**4.实验结果与分析**

**4.1实验设置**

表1真实轨迹数据集

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Sets | Number of Trajectories | Sampling Rates(s) | Points Per Trajectory(k) | Tatal points(M) |
| Taxi | 500 | 60 | ~38.6 | 18.8 |
| UCar | 200 | 3-5 | ~112.0 | 21.9 |
| Geolife | 176 | 1-5 | ~107.5 | 18.5 |
| Mopsi | 51 | 2 | ~121.8 | 7.25 |

**真实轨迹数据集**：我们使用了四个真实轨迹数据集（taxi,Ucar,geolife,mopsi）。Taxi数据集是由北京出租车公司所收集, Ucar是由租车公司所收集,geolife和mopsi分别由geolife和mopsi项目收集。四个数据集采样频率从每秒一个至每分钟一个。

**对比实验**：我们首先对CISED-S与CISED-进行了对比，将两者从压缩率，算法运行时间，平均误差等三个方面进行对比。随后我们对OPLT, LDRH，（m=500）,(m=500)四种算法进行了实验，并从算法运行时间，轨迹压缩率，压缩轨迹平均误差，传输信息量等四个方面进行对比。针对每组数据集，追踪阈值在10到100之间共设置了七组实验，我们所有算法在不同阈值上均进行实验。

**实验环境配置**：为了保证公平性，所有的算法都是用java语言实现，实验环境为 64位PC处理机，4核，i7-6700CPU和16GB内存。

**运行时间：**表示算法的运行时间。

**轨迹压缩率：**保留的压缩轨迹点的个数/原始轨迹中点的个数。

**误差：**原始轨迹中每个轨迹点与压缩轨迹的同步距离取平均值。

**传输信息量：**表示移动对象共向服务器传输了多少的信息量。传输数据包含轨迹点信息和速度信息。其中每传输一个点，包含时间，经度，纬度，消耗24Byte，一次速度包含方向，大小，消耗16Byte。所以传输信息量=传输点个数×24B+传输速度次数×16B。

**4.2实验结果分析**

**4.2.1轨迹压缩算法对比**

**4.2.1.1压缩率分析**

1. 随着阈值的增加，压缩率在降低。

2. CISED-算法压缩效果好于CISED-S算法。在四个数据集上分别是CISED-S的（93.97%，93.96%，81.10%，79.03%）。

**4.2.1.2运行时间分析**

1. 随着阈值的增加，运行时间基本不变。

2. CISED-算法运行时间略高于CISED-S算法，这是由于CISED-在运行过程中需要判别当前点在时刻同步点是否在区域S中。在四个数据集上分别是CISED-S的（115.74%，114.41%，116.30%，123.81%）。

**4.2.1.3平均误差分析**

1. 随着阈值的增加，平均误差在增加。

2. CISED-算法平均误差高于CISED-S算法，这是由于其压缩效果好，保留了更少的点，因此平均误差会更大。在四个数据集上分别是CISED-S的（134.60%，131.83%，128.98%，128.98%）。

**4.2.1.4 总结**

CISED-算法与CISED-S算法运行时间基本一样，得到了更好的压缩效果。同时由于其保留了更少的轨迹点，因此平均误差比CISED-S更大。

**4.2.2轨迹追踪算法**

**4.2.2.1压缩率分析**

1.随着阈值的增加，压缩率在降低。

2.数据集mopsi和geolife拥有较低的轨迹压缩率相比于taxi和UCar，这是由于其采样频率更高。

3.LDRH压缩效果最差，由于其追踪过程的设置过于严格。压缩率较LDRH更低。LDRH算法和算法分别是的（127.06%,252.43%,302.23%,349.42%）和（115.07%,172.36%,193.27%,213.60%）在四个数据集上。OPLT与压缩率基本一样。OPLT是的（100.26%,100.83%,101.06%,101.12%）在四个数据集上。这是由于我们采用圆内接正m边形代替圆，当有一些点落在圆内而不在正m边形中所引起的误差。

**4.2.2.2运行时间分析**

1.随着阈值的增加，LDRH和OPLT算法的运行时间基本不变，而和运行时间增加，这是由于其运行中存在缓冲区。当阈值越大时，缓冲区内轨迹点迭代次数越多，所以运行时间会随着阈值的增加而更长。当阈值为40m时，和运行时间分别是OPLT的（16.38,1.90,7.38,9.51）倍和（24.79,2.82,13.64,18.06）倍在四个数据集上。

2.LDRH,,算法的平均运行时间分别是OPLT的(0.31,0.37,0.35,0.32),(15.24,1.96,8.31,10.12),(23.14,3.02,14.86,18.80)倍在四个数据集上。

**4.2.2.3误差分析**

1.随着阈值的增加，压缩轨迹与原始轨迹之间的平均同步距离在增加。

2.taxi数据集平均误差低于其他三个数据集，这是由于其采样频率高，轨迹压缩率高。保留了更多的原始轨迹点，所以其平均误差会低。

3.轨迹压缩效果越好的算法，平均误差越大，这是由于其保留更少的原始轨迹点。LDRH算法为了保证得到与原始轨迹误差不超过的压缩轨迹，位置跟踪过程阈值设置为。因此其平均误差最低。

4.LDRH,,算法的平均误差是OPLT的 （29.55%,33.49%,27.76%,26.64%），（49.38%,49.86%,50.07%,46.17%）， （96.03%,101.44%,100.16%,99.41%）分别在四个数据集上。

**4.2.2.4传输信息量分析**

1.随着阈值的增加，传输点的信息量在减少。

2. 值得一提的是，在LDRH，，OPLT算法中，所有传输至服务器中的点即为压缩后轨迹的压缩点。而算法中由于存储在服务器缓冲区中的部分点可能会被最后舍去，因此服务器接收到点的数量会多于最后压缩轨迹中点的个数。所以算法虽然轨迹压缩率较低，但是其移动对象向服务器中传输的点的个数较多。

3. LDRH由于其位置跟踪过程阈值，因此在传输最多点的同时传输了最多的速度次数。OPLT算法由于整个过程中存在只更新速度的时候，因此虽然传输点的数量较和少，但是传输速度的次数更多一点。

4.在UCar, geolife, mopsi数据集上，算法LDRH,,, OPLT 传输信息量依次减少。在Taxi数据集上，传输了最多的数据量。这是由于Taxi数据集采样频率较低，因此算法中Predicted part传输了较多的数据，而在Variable part经过验证后只保留了少部分关键轨迹点。

5.LDRH，，算法传输的信息量分别是OPLT的（118.94%,176.62%,197.37%,220.59%）,（107.77%,120.72%,126.28%,134.96%）,（163.44%,154.80%,156.46%,144.47%）分别在四个数据集上。

**4.2.2.5总结**

通过实验我们发现LDRH算法拥有最快的运算时间，但是其传输信息量过多，且其保留的压缩轨迹中点个数最多。算法在LDRH的基础上传输了更少的数据量，提高了压缩轨迹的质量。但是其运算时间大大增加。算法在CDR的基础上，传输数据量和运算时间略微增加，但是其压缩轨迹的质量大大增加，得到了压缩效果很好的压缩轨迹。OPLT算法运行时间较LDRH略有增加，但是相较于和运行时间短很多。在传输更少的信息量的同时，得到了与压缩效果基本一致的压缩轨迹。相较于和，我们的算法运行时空间开销和计算开销更低，且效果更好，因此更适合运用于移动端。

**5.相关工作**

大多数的轨迹跟踪算法，其位置跟踪和轨迹压缩两个过程是同步的，如LDRH，CDR等。而在GRTS算法中，其分离了位置跟踪和轨迹压缩两个过程，由于其轨迹压缩过程可以结合其他轨迹压缩算法，同时对其发送的点再次进行了判断，因此其得到了较好的压缩效果。但是其运行时间及传输信息量等表现较差。我们的OPLT算法，位置跟踪和轨迹压缩是同步的，对接收到的新的轨迹点只进行一次判断，实现了O(n)的时间复杂度，并且大大减少了缓存空间，还得到了压缩效果良好的压缩轨迹。这得益于我们运行过程中三维坐标系的建立与时空锥的判别。

**6.总结**

在这篇文章中，我们总结了LDRH，CDR，GRTS等算法的优缺点。我们首先在CISED-S轨迹压缩算法的基础上进行改进，得到了压缩效果更好的CISED-算法。然后在其基础上，增加速度等信息，改进为OPLT轨迹追踪算法，能够在实现很高轨迹压缩率的同时，减少MO向MOD中传输的信息量。同时其算法运行过程中消耗的内存开销及计算开销都较低，十分适合在移动端运行。并将我们的算法在轨迹追踪系统上进行了运用，得到了良好的实验效果。

轨迹跟踪的研究对于工业具有重要意义，例如快递机器人，我们希望时时获得它的位置信息，因此通过轨迹追踪算法，可以获得它位置信息，当位置偏离了我们所期望的区域，可以通过云端给机器人反馈，调整其运动方向。随着现在各种传感器的广泛运用，未来MO与MOD中交互的信息肯定不仅仅只有轨迹信息，还有考虑轨迹信息与其他信息的结合运用。