|  |
| --- |
| **算法2.1** RANSAC算法 |
| **输入：**数据集合G，模型model，阈值*t*，迭代系数*k* |
| **输出：**环路结果集合model |
| 1. 初始化正确点集合*Q*为空； |
| 1. model 🡨 在G中随机选*n*个点构成初始模型 |
| 1. **while** 迭代次数小于 *k* |
| 1. **for** each *v* ∈ G |
| 1. **if** model(*v*) 正确 |
| 6. do *Q* 🡨 *v*; |
| 7. else |
| 8. continue; |
| 9. **end for;** |
| 10. **end while;** |
| 11. **if** *QL*≠的数目小于*d* |
| 12. model 🡨 在G中随机选*n*个点构成初始模型; |
| 13. **goto 3;** |
| 14. **else** |
| 15. return model, *QL*; |

|  |  |
| --- | --- |
| **算法** | |
| **输入**： | |
| **输出**： | |
| 1. |  |
| 2. |  |
| 3. |  |
| 4. |  |
| 5. |  |
| 6. |  |
| 7. |  |
| 8. |  |
| 9. |  |
| 10. |  |
| 11. |  |
| 12. |  |
| 13. |  |
| 14. |  |
| 15. |  |
| 16. |  |
| 17. |  |
| 18. |  |
| 19. |  |
| 20. |  |
| 21. |  |
| 22. |  |
| 23. |  |
| 24. |  |
| 25. |  |
| 26. |  |
| 27. |  |

|  |  |
| --- | --- |
| **算法** 时空归一化 normalization(trajectorie\_list, ) | |
| **输入**：原始轨迹数据集trajectorie\_list，时间维度权重 | |
| **输出**：三维空间中的轨迹数据集trajectorie\_3d \_list，时空转化因子 | |
| 1. | for trajectory trajectorie\_list |
| 2. | for trajectory |
| 3. | if != null |
| 4. | total\_s += |
| 5. | total\_t += |
| 6. | end if |
| 7. | end for |
| 8. | end for |
| 9. | total\_s / total\_t |
| 10. |  |
| 11. | for trajectory trajectorie\_list |
| 12. | trajectory\_3d=[] |
| 13. | for trajectory |
| 14. |  |
| 15. | trajectory\_3d.append([]) |
| 16. | end for |
| 17. | end for |
| 18. | trajectorie\_3d \_list.append(trajectory\_3d) |
| 19. | **return** trajectorie\_3d \_list, |

|  |  |
| --- | --- |
| **算法3.2 DTW-BDS对应点匹配算法 DTW\_BDS\_match(Q,R)** | |
| **输入**：查询轨迹Q，数据轨迹R | |
| **输出**：R在Q上的DTW-BDS对应点对pair | |
| 1. | pair DTW算法匹配结果存入pair |
| 2. | **for** each in R |
| 3. | **if** 是R的第一个点 |
| 4. | p= BDS(,, DTW().first) |
| 5. | **else if** 是R中间的点 |
| 6. | Q()= BDS(, DTW-BDS(), DTW().first) |
| 7. | **else** |
| 8. | p= BDS(, BDS(), ) |
| 9. | pair (, p) |
| 10. | **end for** |
| 11. | **return** pair |
| 12. |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **算法3.2 DTW-BDS对应点匹配算法 DTW\_BDS\_match(Q,R)** | |
| **输入**：查询轨迹Q，数据轨迹R | |
| **输出**：R在Q上的DTW-BDS对应点对pair | |
| 1. | pair DTW算法匹配结果 |
| 2. | **for** each in R |
| 3. | **if** 是R的第一个点 |
| 4. | Q()= BDS(,, DTW().first) |
| 5. | **else if** 是R中间的点 |
| 6. | Q()= BDS(, DTW-BDS(), DTW().first) |
| 7. | **else** |
| 8. | Q()= BDS(, BDS(), ) |
| 9. | r = |
| 10. | q\_old = pair.getValue() |
| 11. | q\_new = Q() |
| 12. | pair (, Q()) |
| 13. | **while** q\_new.timestamp > q\_old.timestamp and r.pre != null |
| 14. | r = r.pre |
| 15. | q\_old = pair.getValue(r) |
| 16. | q\_new = BDS(r, DTW-BDS(r.pre), DTW-BDS(r.next)) |
| 17. | pair (r, q\_new) |
| 18. | **end while** |
| 19. | **end for** |

|  |  |
| --- | --- |
| **算法** 对应轨迹段时空距离 space\_distance\_calculate(Q, R, pair,) | |
| **输入**：数据轨迹R，查询轨迹Q，R的对应点对pair，断点距离阈值 | |
| **输出**：数据轨迹所有轨迹段到对应轨迹段的时空距离d\_space\_list | |
| 1. | for |
| 2. | if |
| 3. | 使用参数切分轨迹段，获得所有断点 |
| 4. | 从pair中获取和 |
| 5. | 计算断点对应点 |
| 6. | d=，， |
| 7. | w=[1/2, 1, 1…1, , ] |
| 8. | else if |
| 9. | d=， |
| 10. |  |
| 11. | d\_space 🡨 d乘以对应权值w再求和 |
| 12. | d\_space\_list.add(d\_space) |
| 13. | end for |
| 14. | return d\_space\_list |

|  |  |
| --- | --- |
| **算法** 轨迹段形状性计算 sim\_shape\_calculate(,, , ) | |
| **输入**：样本点，及其对应点, | |
| **输出**：轨迹段到对应轨迹段, 的形状相似性sim\_shape | |
| 1. | if or |
| 2. | 计算 |
| 3. |  |
| 4. | return |
| 5. | else |
| 6. | for segment in |
| 7. | = segment[0],segment[1] |
| 8. | 获得的对应点和 |
| 9. | sim\_shape += sim\_shape\_calculate(,, , ) |
| 10. | end for |
| 11. | return sim\_shape |

|  |  |
| --- | --- |
| **算法** 轨迹段形状影响因子计算 I\_shape\_calculate(Q, R, pair, ) | |
| **输入**：查询轨迹Q，数据轨迹R，R的对应点对pair，形状影响因子的权重 | |
| **输出**：数据轨迹所有轨迹段到对应轨迹段的形状影响因子I\_shape \_list | |
| 1. | for in R |
| 2. | sim\_shape= sim\_shape\_calculate(,, , ) |
| 3. | I\_shape= |
| 4. | I\_shape \_list.add(I\_shape) |
| 5. | end for |
| 6. | return I\_shape \_list |

|  |  |
| --- | --- |
| **算法** 局部相似性计算 sub\_similarity\_calculate(Q, R, , , , ) | |
| **输入**：查询轨迹Q，数据轨迹R，激励参数，断点切分阈值，形状权重，子轨迹长度限制参数 | |
| **输出**：Q中与R最相似的子轨迹best\_segment，轨迹距离min\_distance | |
| 1. | pair= DTW\_BDS\_pair\_match(Q,R) |
| 2. | d\_space\_list= space\_distance\_calculate(Q, R, pair,) |
| 3. | I\_shape \_list= I\_shape\_calculate(Q, R, pair, ) |
| 4. | d\_segment\_list = d\_space\_list I\_shape \_list |
| 5. | min\_distance = inf |
| 6. | for R |
| 7. | l = d() |
| 8. | if and <min\_distance |
| 9. | best\_segment = |
| 10. | min\_d= |
| 11. | return best\_segment, min\_distance |