International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 06 Issue: 04 | Apr 2019 www.irjet.net p-ISSN: 2395-0072

Clustering the Real Time moving object adjacent Tracking

Bhupeshwar Kumar Sahu[1], Dr. Vishnu Mishra[2] Dr. Megha Mishra[3]

[1]Ph.D (Research Scholar), Department of IT & CS, Dr. C.V. Raman University, Bilaspur, India[1]
[2]Professor, CSE, Shri Shankarachary Technical Campus, Bhilai, India[2]
[3]Asso. Professor, CSE, Shri Shankarachary Technical Campus, Bhilai, India[3]

Abstract— The moving object trajectory is represented by a polyclone in with respected the time and space each vertices are the represented as any time stamped positions acquired by a suitable positioning system.

The Transmitting and storing every moving object in position an object's trajectory; however, it causes high arbiter cost and communication costs and generally consumes too much storage capacity.

The object moving in former particularly applies if the moving object has adjacent and informed in real-time movement of any object which require storing in sensed information database, as required for many applications. . Many of these systems are based on sensors that are attached to the moving objects. Tracking the trajectories of such objects therefore requires communicating the position data to the moving of data object through over a wireless network.

In this paper, we represented the different trajectory tracking protocols that guided the actual movement of orbiter data . The family is derived from two basic protocols named Connection-Preserving Dead Reckoning (CDR) and sensed information database.

Index Terms— CDR, sensed information database, clustering.

1. INTRODUCTION

In this paper we investigate the computational power of sensor networks in the context of a tracking application by taking a minimalist approach focused on binary sensors. The binary model assumption is that each sensor network node has sensors that can detect one bit of information and Our tracking algorithms have the flavor of particle filter-ing [1] and make three assumptions. First, the sensors across a region can sense the target approaching or moving away. The range of the sensors defines the size of this region which is where the active computation of the sensor network takes place (although the sensor network may extend over a larger area). The second assumption is that the bit of information from each sensor is available in a centralized repository for processing[7,8]. This assumption can be addressed by using a simple broadcast protocol in which the nodes sensing the target send their id and data bit to a base station for pro-cessing. Because the data is a single bit (rather than a complex image taken by a camera) sending this information to the base station is feasible. Our proposed approach is most practical for applications where the target's velocity is slower than the data flow in the network, so that each bit can actually be used in predictions. However, since the accu-racy of our trajectory computation depends on the number of data points, the predictions are not affected by the veloc-ity of the target relative to the speed of communication. The third assumption is that an additional sensor that supplies proximity information as a single bit is available[15,16]. Such a sensor may be implemented as an IR sensor with threshold-ing that depends on the desired proximity range, and can also be derived from the same basic sensing element that provides the original direction bit of information. Line simpli cation refers to a multitude of algorith-mic problems on approximating a given polyline by a simpli ed one with fewer vertices. In the terminology of line simpli cation, trajectory tracking is a min-# prob-lem in R^{1+d} (d = 2 or 3) in the case of Hausdor dis-tance under the (time-)uniform distance metric [3,2]. We investigate two realizations of GRTS with di erent line simpli cation algorithms, namely the op-timal line simpli cation algorithm introduced in [11] and a simple but e cient simpli cation heuristic [18].

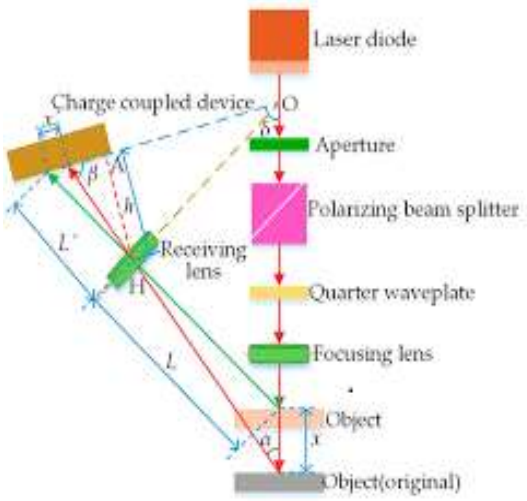

Fig 1: Real Time Sensor Working

2. RELATED WORK

There exist pairs of trajectories that cannot be distinguished by any binary sensor. We con-clude that additional information is needed to disambiguate between different trajectories and to identify the exact loca-tion of the object. This can be realized by adding a second binary sensor capable of providing proximity information (such as

an IR sensor) to each sensor node in the network. If the object is detected within some set range of the prox-imity sensor, that node broadcasts a message to the base station. The range of the proximity sensor may be different and much smaller than the range of the movement direction sensor. It is useful to set the proximity range so that the sen-sors are non-overlapping (this can be done by appropriate thresholding) but this is not necessary. The base station will approximate the location of the object in the region covered by all the sensors reporting object detection. For simplicity of presentation we assume for the rest of the session that the detection range can be calibrated so that at most one sensor detects the object at a time.

To evaluate our approach, we implemented Algorithm 1 in MATLAB and performed extensive simulations on our implementation. All trajectories are taken inside the [0, 1]× [0, 1] square and thus the error measurements are relative to this square. Several types of trajectories have been consid-ered: linear trajectories, trajectories with random turns and trajectories with "mild" turns (at each sensor readings the direction of the tracked object can vary from the previous one with at most $\pi/6$). All trajectories are piecewise lin-ear and the distance traveled by the object between sensor readings is almost constant. A typical simulation example for a linear trajectory (denoted by triangles) can be seen in Fig. 5. The distance traveled between sensor readings is $N(0.12, 0.02)$, i.e. drawn from a normal distribution with a sters obtained from this method may have about 8 times larger radii than the radii obtained by the mean of 0.12 and a standard deviation of 0.02.optimal clustering, and the numbers of clusters are also much larger (at least 15 times) than for the usual clustering. Further, this proposal does not take into account I/O efficiency.

The sensed trajectory s(t) generally deviates from a(t) due to inaccuracies of the positioning sensor and the time-discrete sensing. The former are generally de-scribed by stochastic means such as probability density functions or percentiles, which allow deriving a maxi-mum sensor inaccuracy that holds with high prob-ability. Inaccuracies beyond (typically indicated by erratic positions) are considered as errors. They have to be treated separately, e.g., by informing the MOD that there will be no valid trajectory information un-til further notice. Regarding the time discretization by position sensing, the movement between two sensing operations is subject to physical constraints like the maximum speed or acceleration.

3. MOVING-OBJECT SENSOR BESED ALGORITHM

This section first describes the representation of moving ob - jects, then proposes a scheme to cluster moving objects, called Moving-Object Clustering (MC for short).

ID OID can be represented by a four-tuple (OID , $\bar{x}_u$, $\bar{v}$, t_u), where $\bar{x}_u$ is the position of the object at time t_u and $\bar{v}$ is the velocity of the object at that time.

To facilitate the analysis, we initially assume that no updates occur to the dataset. This enables us to set the weights used in M to 1—decreasing weights are used to make later positions, which may be updated before they are reached, less important. Also to facilitate the analysis, we replace the sum of sample positions in with the corresponding integral, denoted as *M ′, from the time when a clustering is performed and U time units into* the future. Note that M ′ is the boundary case of M that is similar to the integrals used in R-tree based moving object indexing [21].

The next theorem states that inclusion of an object into the cluster with a smaller M ′ value leads to a tighter and thus better clustering during time interval U . moving object wants to stop reporting its movement, it sends a nal update message with the most recent sensed position { but without a new pre-diction { and terminates the algorithm (line 18).

Sensor motion DB Algorithm (0uID$_1$, O, 0uID$_2$)

Input: Cluster ouID$_1$ and object O1

Result: New cluster with IoD OuID2

4: U U k (last(S))

5: S (last(S))

6: end if

7: U U k (last(S))

8: S (last(S))

9: end if

10: S S k (s_R) . Append s_R to sensing history.

11: if LDR causes update then

12:: U U k (last(S)) . Append s_R as u_n.

compute new predicted velocity . . .

if $D_{m1} > D_{m2}$ **then**

insert O_r into cluster OuID$_2$

modify the hash table

11: v **if** O_r belongs to cluster OuID$_1$ **then**

12 :send update message (jV n Uj; U n V; $_V$) to

13:: S S k (s_R) . Append s_R to sensing history.

14: if LDR causes update then

15: U U k (last(S)) . Append s_R as u_n.

16: V compute new predicted velocity . . .

17:send update message (jV n Uj; U n V; V) to **for** each remaining object O_r in CID_1 **do**

$D_1 \leftarrow M(O_r, move_1)$

$D_2 \leftarrow M(O_r, move_2)$

18: removeO_r from cluster $OuID_1$adjust the clustering feature of cluster $OuID_1$

19: Anysis and computing clustering feature of cluster $OuID_2$

return CID_2

end

Fig. 2. Sensor Motion DB Algorithm

After a DB Clustering , we check whether each cluster C among the two new clusters can be merged with preexisting clusters (see Figure 8). To do this, we compute the M -distances between the center object of cluster C and the center object of each preexisting cluster. We consider the k nearest clusters that may accommodate cluster C in terms of numbers of objects. For each such candidate, we execute a "virtual merge" that computes the clustering feature assuming absorption of C. This allows us to identify clusters where the new average radius is within threshold ρ_g .

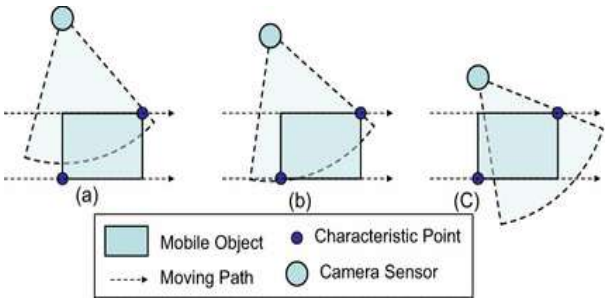

Fig 3: Moving Trajectory object

Delete (O**)**

Input: O is an object to be deleted

1. CID = Hash (O)

   // object O belongs to cluster CID

2. delete O from the hash table

3. delete O from cluster CID

4. adjust the clustering feature of cluster CID

5. **if** cluster CID is in underflow

6. **if** CanMerge(CID , CID ′)

7. **then** merge(CID , CID ′)

8. **else**

9. delete old event of cluster CID from the event queue

10. insert new event of cluster CID into the event queue end Delete.

Fig. 3. Deletion Algorithm

from the hash table and cluster C, and we adjust the clustering feature. Specifically, we first update the feature to the curr ent time according to Claim 1 and then modify it according to Claim 2. If cluster C does not underflow after the deletion, we further check whether the split event of C has been affected and adjust the event queue accordingly. Otherwise, we apply the merge policy to determine whether this cluster C can be merged with other clusters (denoted as C I D′). The deletion algorithm is outlined in Figure 3.

3) *Split and Merge of Clusters:* Two situations exist where a cluster must be split. The first occurs when the number of obje cts in the cluster exceeds a user-specified threshold (i.e., the maxi-mum cluster capacity). This situation is detected automatically by the insertion algorithm covered already. The second occurs when the average radius of the cluster exceeds a threshold, which means that the cluster is not compact enough. Here, the threshold (denoted as ρ_s) can be defined by the users if they want to limit the cluster size. It can also be estimated as the average radius of

√
—

clusters given by the equation $\rho_s = \frac{1}{4} S_c$. We proceed to address the operations in the second situation in some detail.

Recall that the average radius of a cluster is given as a function of time R(Δt) (cf. Section III-C). Since R(Δt) is a square root, for simplicity, we consider $R^2(\Delta t)$ in the following computation. Generally, $R^2(\Delta t)$ is a quadratic function. It degenerates to a linear function when all the objects have the same velocities. Moreover, $R^2(\Delta t)$ is either a parabola opening upwards or an increasing line—the radius of a cluster will never first incr ease and then decrease when there are no updates. Figure 4 shows the only two cases possible for the evolution of the average radius when no updates occur, where the shaded area corresponds to the region covered by the cluster as time passes.

Our task is to determine the time, if any, in-between the current time and the maximum update time when the cluster must be split, i.e., t ranges from 0 to U . Given the split threshold $\rho_s$relationships between $R^2(\Delta t)$ and ρ^2_s are possible—see Figure 5.

The next step is to identify each of the three situations by means of function $R^2(\Delta t)$ itself. We first compute $R^2(0)$. If this value exceeds ρ^2_s, we are in the second case. Otherwise, $R^2(U)$ is computed. If this value is smaller than ρ^2_s, we are in the first case. If not, we are in the third case, and we need to solve the equation $(A\ t^2 + B\ t + C)/N = \rho^2_s$, where the split time t_s is

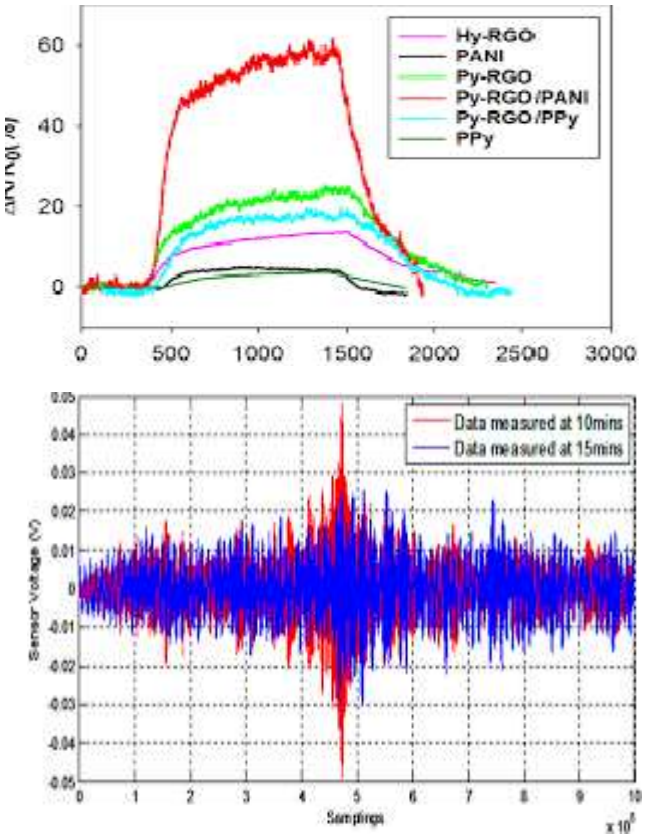

Fig 4: Sensor object moving tracking adjacent result

At the time of a split, the split starts by identifying the pair of objects with the largest M value. Then, we use these objects as seeds, redistributing the remaining objects among them, again based on their mutual M values. Objects are thus assigned to the cluster that they are most similar to. We use this splitting procedure mainly because it is very fast and running time is an important concern in moving object environments. The details of following.

Tracking Stutus	Risk of movement	Position in Accuracy	New position
Accuracy	<1.0	average	yes
No accuracy	=0.97	average	average
Defective ness in accuracy	==0.65	average	no
Actual position in accuracy	==0.78	High	yes
New position accuracy	>1.0	High	Yes

Table 1: Accuracy of Sensor DB of moving Object

4. CONCLUSIONS

In this paper, we presented the Connection-Preserving Dead Object moving and Trajectory object motion protocols for find the motion of orbital object tracking the trajectories of moving objects with embedded positioning sensors at a remote moving of elemental attributes.

For fulfilling such aim, the moving can sense by objects can sense their positions periodically but report only a subset of the positions to the MOD so that the resulting simpli ed trajectory approximates the actual movement according to a pre-de ned accuracy bound. To inform the MOD about the current position, CDR and GRTS use dead reckoning.

CDR is solely based on dead reckoning whereas GRTS separates the tracking of the current position from the simply caption of the past trajectory. Therefore, GRTS outperforms CDR by more than a factor

REFERENCES

[1] M. A. Abam, M. de Berg, P. Hachenberger, and A. Zarei. Streaming Algorithms for Line Simpli cation. In Proc. of 23rd Symp. on Computational Geometry (SCG), pages 175{183, Gyeongju, South Korea, 2007.

[2] P. K. Agarwal, S. Har-Peled, N. H. Mustafa, and Y. Wang. Near-Linear Time Approximation Algorithms for Curve Simpli cation. Algorithmica, 42(3{4):203{219, 2005.

[3] D. Crisan and A. Doucet. A survey of convergence result on particle filtering for practitioners, 2002.

[4] H. Cao, O. Wolfson, and G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds. VLDB Journal, 15(3):211{228, 2006.

[5] W. S. Chan and F. Chin. Approximation of Polygonal Curves with Minimum Number of Line Segments. In Proc. of 3rd Int'l Symp. on Algorithms and Computation (ISAAC), pages 378{387, Nagoya, Japan, 1992.

[6] R. R. Brooks, P. Ramanathan, and A. Sayeed, DistributedTarget Tracking and Classification in SensorNetworks, *Proceedings of the IEEE*, September 2002.

[7} R. H. G•uting and M. Schneider. Moving Objects Databases. Morgan Kaufmann, San Francisco, CA, 2005.

[8} J. Hershberger and J. Snoeyink. An O(n log n) Implemen-tation of the Douglas-Peucker Algorithm for Line Sim-pli cation. In Proc. of 10th Symp. on Computational Geometry, pages 383{384, Stony Brook, NY, 1994.

[9] B. Krishnamachari, Energy-Quality Tradeoffs for Target Tracking in Wireless Sensor Networks, *IPSN 2003*, 32-46.

[10] D. Salmond, N. Gordon and A. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE Proc.F, Radar and signal processing*, 140(2):107–113, April 1993.

[11] R. Lange, F. D•urr, and K. Rothermel. Online Trajectory Data Reduction using Connection-preserving Dead Reck-oning. In Proc. of 5th Int'l Conf. on Mobile and Ubiqui-tous Systems (MobiQuitous), Dublin, Ireland, 2008.

[12] W.E.L. Grimson, C. Stauffer, R. Romano, and L. Lee. Using adaptive tracking to classify and monitor activities in a site. In *Proc. of IEEE Int'l Conf. on Computer Vision and Pattern Recognition*, 22–29, 1998.

[13] Lynne E. Parker. Cooperative motion control for multi-target observation. In *Proc. of IEEE International Conf. on Intelligent Robots and Systems*, pages 1591–7, Grenoble, Sept. 1997.

[14] P. Misra and P. Enge. Global Positioning System: Signals, Measurements and Performance. Ganga-Jumuna Press, 2001.

[15] M. F. Mokbel, T. M. Ghanem, and W. G. Aref. Spatio-Temporal Access Methods. IEEE Data Engineering Bulletin, 26(2):40{49, 2003.

[16] M. Potamias, K. Patroumpas, and T. Sellis. Amnesic On-line Synopses for Moving Objects. In Proc. of 15th ACM Int'l Conf. on Information and Knowledge Management (CIKM), pages 784{785, Arlington, VA, 2006.

[17] J. Rankin. GPS and Di erential GPS: An Error Model for Sensor Simulation. In Position Location and Navigation Symp., pages 260{266, 1994.

[18] D. Tiesyte and C. S. Jensen. Recovery of Vehicle Trajectories from Tracking Data for Analysis Purposes. In Proc. of 6th European Congress and Exhibition on Intelligent Transport Systems and Services, Aalborg, Denmark, 2007.

[19] U.S. Dept. of Defense. Global Positioning System Standard Positioning Service Performance Standard, 2001.