分散協調型多体軌跡推定システムの高機能化

山本孝一*・下川俊彦*
吉田紀彦**

Improvement of Cooperative System for Multi-Target Motion Analysis

by

Koichi YAMAMOTO*·Toshihiko SHIMOKAWA*
and Norihiko YOSHIDA**

Track estimation of targets from sensor data is a crucial issue in active dynamic scene understanding. Multitarget motion analysis, where there are multiple moving targets and multiple fixed sensors which only measure bearings of the targets, is to associate targets and sensor data, and estimate target tracks based on that association. This is an NP-hard problem in general, and solved using stepwise relaxation. However, it is hard to obtain the optimal solution, as the method easily gets trapped in one of local optima.

We applied the decentralized cooperative search technique to this problem, and proved our method effective. The method uses more than one processors, each of which has its own partial search space, searching multiple possibilities in parallel. This approach leads to cooperative distributed vision. We are currently extending our method to address the case where targets move toward varying directions and in varying velocities. This paper presents the current status of our research, and presents two prototypes of cooperative multiagent systems for extended multitarget motion analysis.

1．はじめに

近年，方位角のみを検出するセンサからの情報を利用して，移動する物体の軌跡を推定しようとする研究が幾つかなされている。また，方位角を検出できるセンサの研究も行なわれている。

これまでの軌跡推定の研究は，移動する1つのセンサからの方位角情報を元に，等速運動をする1つの物体の軌跡を推定する単体軌跡推定がほとんどであったが，文献1）ではこれを複数の物体に対する多体軌跡推定に拡張を試みている。

この手法は固定された複数のセンサからの方位角情報を統合して，等速運動をする複数の物体の移動軌跡を推定するものである。

このように複数のセンサからの情報を統合して適切な解を求めるシステムは分散センサネットワーク（Distributed Sensor Networks 略して DSN）と呼ばれる。

これは分散人工知能の分野に含まるもので，数多くの研究がなされている[2)]。なかでも通信戦略や情報統合の統合アルゴリズムは主な研究テーマとなっており，DVMT（Distributed Vehicle Monitoring Testbed[3)]）はその例題として有名である。

平成11年4月23日受理

*九州大学大学院システム情報科学研究科　福岡市東区箱崎
（Graduate School of Information Science and Electrical Engineering, Kyushu University, Hakozaki, Fukuoka）
**情報システム工学科（Department of Computer and Information Sciences）

1）で提案された多体軌跡推定の手法は，全てのセンサからの情報を1つの問題解決PEに集めて集中的に処理するものである。

推定にあたっては最尤法を用いて解を探索している。しかしこの問題において，解空間には無数の局所解が存在している。そのため局所解に頻繁に陥ってしまい，最適解が得られにくいということがこの手法の大きな欠点である。それを改善するために，1）では確率的探索である焼鈍法（Simulated Annealing）を導入しているが，その結果処理時間が大幅に増大してしまった。一般に分散処理化することによる利点として解の多様性が増大することにより局所解に陥る危険が減少し，処理の分散化により処理速度が向上することがいわれている。

論文4）では1）の手法を基にして，システム内に複数の問題解決PEを存在させ，PE 同士が通信により協調を計る分散協調処理を提案した。

また実際には，最初からデータが揃っているわけではなくデータは一つずつ逐次的に入力される。現実の世界においてこのようなシステムを使うためにはこのような入力に対応できるような入力のシステムが必要である。

以下では第2章で多体軌跡推定の問題を形式化し，第3章で従来の集中処理による推定及び焼鈍法を含む手法を紹介する。第4章で分散協調処理の手法を提案し，第5章で実時間処理システムへの改良およびその実装について述べ，第6章ではそれらの手法と従来の手法についてそれぞれ実験並びに比較を行ない，最後に第7章でまとめる。

2．多体軌跡推定問題

図1に示すように，システムは等速運動をするN個のターゲットと，ターゲットの方位角を検出できる固定されたs個のセンサからなる。各センサは時刻毎の方位角データを得る。多体軌跡推定とは，これら複数のセンサからのデータを統合して各ターゲットの初期状態（r_x, r_y, v_x, v_y）を求めるものである。

ここで問題となるのは，

- センサの得たデータとターゲットとの対応をどのようにとるか
- 対応をとったデータから軌跡をどのように推定するか

の2つである。

これまでシステムの概略を述べたが，これらの問題をを実際に形式化する。ターゲットの状態は位置（r_x^t, r_y^t）と速度（v_x^t, v_y^t）から次のように定義できる。

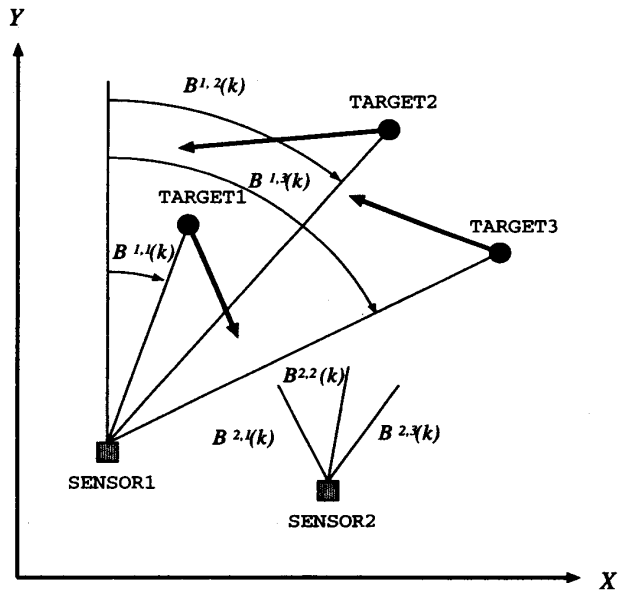

図1　システムの概要

$$X^t(k)=\begin{pmatrix} r_x^t(k) \\ r_y^t(k) \\ v_x^t(k) \\ v_y^t(k) \end{pmatrix},\ t=1,\ \cdots,\ N \tag{1}$$

ここで

$$\begin{aligned} r_x^t(k) &= r_x^t(0)+k\Delta v_x^t(0) \\ r_y^t(k) &= r_y^t(0)+k\Delta v_y^t(0) \\ v_x^t(k) &= v_x^t(0) \\ v_y^t(k) &= v_y^t(0) \end{aligned} \tag{2}$$

tはターゲット識別子，Δはサンプリング間隔，kは時刻識別子を表す。センサの状態は位置（r_{xs}^i, r_{ys}^i）より

$$X_s^i=\begin{pmatrix} r_{xs}^i \\ r_{ys}^i \\ 0 \\ 0 \end{pmatrix},\ i=1,\ \cdots,\ S \tag{3}$$

と定義できる。ここでiはセンサ識別子を表す。センサiから見たターゲットtの相対的な状態ベクトルは

$$X^{t,i}(k)=X^t(k)-X_s^i=\begin{pmatrix} r_x^{t,i}(k) \\ r_y^{t,i}(k) \\ v_x^t(k) \\ v_y^t(k) \end{pmatrix} \tag{4}$$

時刻kにセンサiがターゲットtに関して観測する方位角データは

$$\beta^{t,i}(k)=\tan^{-1}\left[\frac{r_x^{t,i}(k)}{r_y^{t,i}(k)}\right]+\nu^{t,i}(k) \tag{5}$$

となる。ここで $v^{t,i}(k)$ はセンサ i における白色ノイズである。以上より時刻 k にセンサ i が観測する観測ベクトル $\beta^i(k)$ は式(5)より

$$\beta^i(k)\triangleq\begin{pmatrix}\beta^{1,i}(k)\\ \cdots\\ \beta^{N,i}(k)\end{pmatrix} \tag{6}$$

となる。また，時刻 k までの観測ベクトルの累積 β^k を

$$\beta^k\triangleq(\beta^1(1)',\cdots,\beta^1(k)',\cdots,\beta^s(1)',\cdots,\beta^s(k)')' \tag{7}$$

と定義する。

センサの観測した観測ベクトルには，それぞれの観測値がどのターゲットに対応しているかというデータが含まれていないので，観測ベクトル $\beta^i(k)$ に 0－1 のみからなる $N\times N$ の割り当て行列 $C^i(k)$ を乗じてやる必要がある。この $C^i(k)$ は $N!$ 通り存在する。例えば $N=3$ ならば $C^i(k)$ は以下の 6 通りが存在する。

$$C^i(k)=\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix},\ \begin{pmatrix}1&0&0\\0&0&1\\0&1&0\end{pmatrix},\ \begin{pmatrix}0&1&0\\1&0&0\\0&0&1\end{pmatrix},$$
$$\begin{pmatrix}0&1&0\\0&0&1\\1&0&0\end{pmatrix},\ \begin{pmatrix}0&0&1\\1&0&0\\0&1&0\end{pmatrix},\ \begin{pmatrix}0&0&1\\0&1&0\\1&0&0\end{pmatrix} \tag{8}$$

$[C^i(k)]_{jt}=1$ は観測ベクトル $\beta^i(k)$ の j 番目の要素がターゲット t に対応することを表す。例で示す。

$$ex)\ \begin{pmatrix}0&1&0\\0&0&1\\1&0&0\end{pmatrix}\cdot\begin{pmatrix}\beta^{1,i}(k)\\ \beta^{2,i}(k)\\ \beta^{3,i}(k)\end{pmatrix}$$
$$\Rightarrow\begin{pmatrix}\beta^{3,i}(k)\\ \beta^{1,i}(k)\\ \beta^{2,i}(k)\end{pmatrix}\begin{matrix}\leftarrow\text{correspond to the target 1}\\ \leftarrow\text{correspond to the target 2}\\ \leftarrow\text{correspond to the target 3}\end{matrix}$$

割り当て行列の集合：

$$C^k\triangleq\{C^1(1),\cdots,C^1(k),\cdots,C^s(1),\cdots,C^s(k)\} \tag{9}$$

が全観測データのターゲットに対する組み合わせを決定する。

システム全体は式(10)で記述できる。

$$X^{t,i}(k+1)=\Phi X^{t,i}(k),t=1,\cdots,N,i=1,\cdots,s \tag{10}$$

ここで

$$\Phi=\begin{pmatrix}1&0&\Delta&0\\0&1&0&\Delta\\0&0&1&0\\0&0&0&1\end{pmatrix}$$

は状態遷移行列を表す。

3．集中処理型の推定

全てのセンサの情報を 1 つの PE に集めて集中的に処理する参考文献 1）の手法を述べる。

ターゲットの推定初期状態

$$\hat{X}_0=\begin{pmatrix}\hat{X}_0^{1'}\\ \vdots\\ \hat{X}_0^{N'}\end{pmatrix},\ t=1,\cdots,N, \tag{11}$$

ここで

$$\hat{X}_0^t=\begin{pmatrix}\hat{r}_x^t(0)\\ \hat{r}_y^t(0)\\ \hat{v}_x^t(0)\\ \hat{v}_y^t(0)\end{pmatrix},\ t=1,\cdots,N, \tag{12}$$

が与えられると，センサ s が時刻 j に得るターゲット t に関する方位角データは

$$\hat{\beta}^{t,i}(j,\hat{X}_0)=\tan^{-1}\left[\frac{\hat{r}_x^{t,i}(j)}{\hat{r}_y^{t,i}(j)}\right]. \tag{13}$$

と推定できる。センサ s の時刻 j における推定観測ベクトルは式(13)より

$$\hat{\beta}^i(j,\hat{X}_0)=\begin{pmatrix}\hat{\beta}^{1,i}(j,\hat{X}_0)\\ \vdots\\ \hat{\beta}^{N,i}(j,\hat{X}_0)\end{pmatrix}, \tag{14}$$

となる。また時刻 k までの推定観測ベクトルの累積は

$$\hat{\beta}^k(\hat{X}_0)\triangleq(\hat{\beta}^1(1,\hat{X}_0)',\cdots,\hat{\beta}^1(k,\hat{X}_0)',\cdots,$$
$$\hat{\beta}^s(1,\hat{X}_0)',\cdots,\hat{\beta}^s(k,\hat{X}_0)')' \tag{15}$$

実際のターゲットの軌跡に対する推定した初期状態の確かさの評価に平均二乗誤差（ASE）を用いる。

$$\Lambda(\beta^k|C^k,\hat{X}_0)=$$
$$\frac{1}{c}\,exp\left(-\frac{1}{2}\cdot\sum_{i=1}^{s}\sum_{j=1}^{k}[\beta^i(j)-C^i(j)\cdot\hat{\beta}^i(j,\hat{X}_0)]'\right. \tag{16}$$
$$\left.\cdot R_i^{-1}[\beta^i(j)-C^i(j)\cdot\hat{\beta}^i(j,\hat{X}_0)]\right).$$

$$E=\frac{1}{skN}\sum_{i=1}^{s}\sum_{j=1}^{k}[\beta^i(j)-C^i(j)\cdot\hat{\beta}^i(j,\hat{X}_0)]'$$
$$R_i^{-1}[\beta^i(j)-C^i(j)\cdot\hat{\beta}^i(j,\hat{X}_0)] \tag{17}$$

ここで $R_i=\sigma_i^2 I$ はセンサ i におけるノイズの共分散行列($N\times N$)である。E とターゲットの関係を考慮して式(17)を書き換えると

$$E=\frac{1}{skN}\sum_{t=1}^{N}E_t$$
$$=\frac{1}{skN}\sum_{t=1}^{N}\sum_{i=1}^{s}\sum_{j=1}^{k}\left(\frac{\beta^{c(i,j,t),i}-\hat{\beta}^{t,i}}{\sigma_i}\right)^2 \quad (18)$$

となる。ここで $[C^i(j)]_{mt}=1$ は $c(i,j,t)=m$ を表す。

多体軌跡推定の問題は，この平均二乗誤差（ASE）を最小化する問題であると言い替えることができる。まず割り当て行列 C^k を固定した場合，Gauss-Newton 法を用いて $\hat{X}^t_0$ を決めることで E_t を最小化することができる。つまりターゲット t について並べられた方位角のデータ列からターゲットの初期状態 $\hat{X}^t_0$ を計算することができる。また，式(18)は非負の数の加算であるので，$\hat{X}_0$ が与えられた場合に C^k に関して E を最小化することは，$C^i(j)$ に関して個々の項を最小化することに等しい。よって次の操作を行なうことで，初期状態 $\hat{X}^t_0$ を固定した場合に E_t を最小化する $C^i(j)$ を求めることができる。

For all $C^i(j), i=1, 2, \cdots, s, j=1, 2, \cdots, k$

$$\min_{C^i(j)}\left[\beta^i(j)-C^i(j)\cdot\hat{\beta}^i(j,\hat{X}_0)\right]'$$
$$R_i^{-1}\left[\beta^i(j)-C^i(j)\cdot\hat{\beta}^i(j,\hat{X}_0)\right] \quad (19)$$

Next i, j.

このアルゴリズムにおいて $\hat{X}^l_0$ と $C^{k,l}$ は l 回目の繰り返しでの $\hat{X}_0$ と C^k の解である。Step 1）とStep 2）の両段階において E は単調減少となるので，この繰り返しで E は局所的な最小値には到達することが保証される。しかし，この解空間には局所解が多数存在しており，この手法ではかなりの場合に局所解に陥ってしまう。そこでこれを避けるために焼鈍法を含む手法が提案された。

4．分散協調による改良

4.1 分散処理化

分散処理化にあたり，システム内の全センサを複数のグループに分割する。分割されたセンサ群は1つのPE に属すことになり，それらのデータの処理は各PE が受け持つ。1つのセンサが複数のセンサ群に含まれることも許す。

また，グループ分けすることにより解に多様性が出て，局所解に陥る可能性が減るという利点もある。また，これらの中間解を交換することにより，協調的な処理が実現できるので，効率的なシステムの構築が期待できる。

4.2 通信戦略

4.2.1 分散孤立型

それぞれのグループがお互いに通信を行なわない最も簡単な分散処理の手法。各 PE は集中処理型と同様の反復を繰り返すもので，集中処理型のものが複数個集まっただけのものと考えられる。

各 PE はそれぞれのローカル ASE が最小になるように軌跡を推定する。通信を行なわないので，他の PE での結果は軌跡推定に全く反映されない。しかし集中処理型と比べると，センサ群が複数存在するので解の多様性が得られる。また，処理を分散しておこなっているので処理コストも軽減している。

4.2.2 ローカル法

ローカル ASE に基づく通信戦略である。1回の軌跡推定が終了する毎に各 PE はその時点の推定初期状態を通信により交換し合う。PE はそのようにして得た推定初期状態から自分のセンサ群におけるローカル ASE を計算する。結果として PE はシステム内のグループ数だけローカル ASE を得るが，その中で最もローカル ASE が小さいものを次の反復における初期状態として採用する。つまり反復毎に各 PE は複数の初期状態の候補の中から自分達のグループにとって最も都合のよい初期状態を選んで採用していく。システム全体としての利益は考慮に入れておらず，それぞれのグループは利己的に働くといえる。しかしグループが複数存在するので，解の多様性が得られる。

4.2.3 グローバル法

グローバル ASE に基づく通信戦略である。1回の軌跡推定が終了する毎に各 PE はその時点の推定初期状態におけるグローバル ASE を計算する。PE は自分の推定初期状態とそれから得たグローバル ASE を通信により交換する。通信の結果各 PE はシステム内のグループ数だけ初期状態と，その時のグローバル ASE を得る。その中から最もグローバル ASE が小さいものを選んで次の反復の初期状態として採用する。結果としてシステム全体から見て最も適した初期状態が，全てのグループで一様に採用される。このようにすることで反復毎に各 PE で局所解を避けて初期状態が選ばれることになる。通信により各グループはシステム全体としての目的を考慮して振舞う。

前述のような方法で推定を進めていくが次に各プロセッサの推定処理が終了するための条件を以下に示す。

- 各プロセッサの推定処理の終了条件は，（ASEの

変化）$<10^{-5}$とする。それ以降，そのプロセッサは推定しない。
- その他のプロセッサは，終了したプロセッサの推定結果を引続き利用する。
- 全プロセッサが推定し終えた時を終了とする。この時，各プロセッサの ASE と比較して，最小 ASE を出したプロセッサのデータを最終推定結果とする。

今までのシミュレーションによる実験では，データの格納は，多次元配列を用いていたため，データを上書きしない限りそのデータはいつでも参照可能であった。さらに，通信を必要とせず，配列参照でデータ交換が可能なので，送受信のタイミングなども気にする必要もなかった。しかし，実際の分散環境下では，通信によるデータのやりとりを実現する場合，送信側と受信側がそれぞれ然るべき処理をとらなければならない。具体的にはシミュレーションではデータが欲しい側がその部分の配列を参照するだけで参照される側は何の処理も必要なかったが，分散化されると参照される側がまずデータを送信し，参照する側がそのデータを受信するという処理手順になる。

実際の分散環境下では，いつ他のプロセッサが終了したのかが見えないので，他のプロセッサが終了したにも関わらず，データを待っている状態になる可能性がある。そのためには他のプロセッサの状況を把握して，それに応じた処理をしなければならない。そこでプロセッサの状態を示す flag を導入し，flag を送受信することによって，他のプロセッサの状態を把握させるようにした。具体的には break flag と iteration flag があり，前者はプログラムを終了すべきかどうかを判定する flag で，後者はそのプロセッサでの計算が終了したかどうかを示す flag である。

サーバが，クライアントから iteration flag を受けとることによって，各プロセッサの状態を知ることが出来，全てのプロセッサの計算が終了した時クライアントに break flag を送信し，プログラムを終了する。

5．実時間処理システムへの改良

5.1 実時間処理システムへの移行

実時間処理システムというのは，次々と入力データが入り，それらを次々と裁いていき，解を求めるようなシステムのことである。

これを多体軌跡推定で実現しようというのが，今回のシステムである。具体的には，次々と得られる，固定された複数のセンサからの方位角情報を統合して，等速運動をする複数の物体の移動軌跡を次々と推定するシステム，と再定義される。

その概要を下に述べる。
- このシステムへの改良点を少なくして拡張する形で実装する方法が望ましい
- 入力が間欠的になり従来より多くのデータを連続的に扱わねばならない。
  - 連続的に入ってくるデータの中から前システムと同数のデータを選びだし推定を行なうという形で実装
- このシステムの処理速度が入力データに追い付かない場合，現在の処理時間を短縮しなければならない可能性もある。

これらを考慮しながら，実装を進めて行く必要がある。

5.2 分散型実時間処理の入力システムについて

入力はセンサによって行なわれる。その過程は次のようになる。各センサの入力はグルーピングにしたがって，入力データを分散し，割り当てられたプロセッサに入力される。それに伴い，各プロセッサは入力されたデータでそれぞれ推定を開始する。これまでのシステムと今回の実時間処理システムの入力の違いについて述べる。

今までは，データは一つづつ入力されるのではなく，一定数の入力データをまとめてシステムに入力していた。よって，実時間処理システムを実現するに当たっては今回もバッチ処理によるシミュレーションでシステムを構築した。しかし，実時間処理を実装する場合，推定処理を進めながら次々と入力されるされるデータを扱わねばならない。将来的にセンサの問題や入力データの同期に対応できるようなモデルを考案しなければならない。

まず，センサから送られてきたデータを入力システムに蓄積する。この data pool から一度の推定に PE が使うデータの個数のみを入力する。そのデータを選ぶ関数についてであるが，以下のようないくつかのアイディアが考えられる。

1．単純にデータの個数を増やす（10→100→1000→…）。これは推定回数を減らすことが出来る。
2．n 個データがたまるごとに推定を行ない，処理を終えると次の n 個で推定を行なう（隣接した推定間でのデータの重なりがない）。具体的にはデータをためる部分と計算部分を分け，前者はデータを n 個ずつに区切りながらため続け，後者はその区切られた n 個ずつのデータで推定を行ない続ける。これにより入力されたデータを無駄にすることがなく，正確な推定が可能になる。

3．queue のように s 個だけ新しいデータを入れ，古い順に s 個データを破棄して推定を繰り返す方法である。この方法の特徴としては，

・s の履歴を記憶しておいて，動的に次回の s の値を決定する方法も考えられる。

各方法の主な短所は以下の通りになる。

1．計算量が莫大になる。

2．この方法は入力の頻度が高い場合に処理が入力に追い付かない。

3．入力データを無駄にする可能性がある。

上の3つの中で無視できないのは，1と2の方法であろう。この2つはいずれもシステムのパフォーマンスを下げることになる。しかし3の短所は，このシステムの性質を考えると重要ではないと考えられる。従って今回は3の方法を採用した。以下に，この方法での具体的な実装方法について述べる。

まず，n については現システムの入力と同数で考慮。次に s については，k 回目の推定と（k+1）回目の間に新たに入力された数を指定している。すなわち，各推定毎に新しく入力されたデータ数だけずらすということになる。実際には，n 個のキューを用意し，新たにデータが入力される度にキューイングしていく。この時排出されたデータは使わない。要するに時系列に並んだ最新のデータの内，新しい順に n 個を選ぶということである。この方法により入力データの数によらず確実に一定個数の最新データを確保できる。そのアルゴリズムを疑似言語で表現すると以下のようになる。

要素数 n の配列を用意する。

```
if new__data
for(i= 0 ; i<n ; i++)
begin
  data[k]=data[k+1] ;
  data[max__data]=new__data;
end
```

5.3 システムの実装

5.2節で，入力システムの構成について述べた。この構成を実装レベルで書き下すと，

1．最初に初期位置と速度ベクトルを入力させ，これを元に予めサンプリングを作る。

2．それらの中から data を選びだしシステムに入力 n,s の値についてはプログラムの始めに入力するようにしている。そして，推定毎に s の分だけずらしたデータをシステムへの入力とした。

そのアルゴリズムを疑似言語で表現すると，

```
for(j= 0 ; j<num__of__sensor; j++)
  for(i= 0 ; i<num__of__target; i++)
    for(t= 0 ; t<n; t++)
      sensor. B[i][t]=sensor. sample__B[i][t+s];
```

で表現できる。ここで num__of__sensor はセンサの数，num__of__target はターゲットの数とする。

ただし今回実装したシステムは実時間的に一つずつ入力される形式ではなく，各推定間の入力が s 回存在したと仮定するシミュレーションである。従って，理論的なアルゴリズムとは入力のタイミングが異なる。

6．実験と評価

6.1 シミュレーションとの比較実験

6.1.1 分散環境での実験

まず，論文4）でのシミュレーションと今回実装した実際の分散環境でのシステムで性能比較した。但し，前節で述べた，終了条件は2つのプログラムで実際に分散化されたかどうかを除いて全く同じにするために変更した．

その際，ターゲットを4個センサを3個としてセンサを配置した。

ここでは，1つの例についての実験を行なった。この実験でのサンプリング数もシミュレーションと同じ条件にするため，100とした。次に推定結果を表すと図2のようになる。ここで，ASE の推移は図3のようになった。

また，その推定軌跡を図4に示す。これらにより実際に分散化しても，性能を落すことなく推定可能で，その速度比は2.44であることが分かった。このことは論文4）において提唱されていたが，分散協調処理が多体軌跡推定において，非常に有効であることを改めて示している。その理由としては分散処理化により処理能力が向上し，局所解に陥る危険性が減ったことが挙げられる。また PE 同士が協調することでさらに精度が向上していることも分かる。

図2 シミュレーションと分散環境の比較

	Simulation	Distributed
Final ASE	0.0	0.0
Time	52.07	21.375
速度比	1	2.44
Iteration	9	9

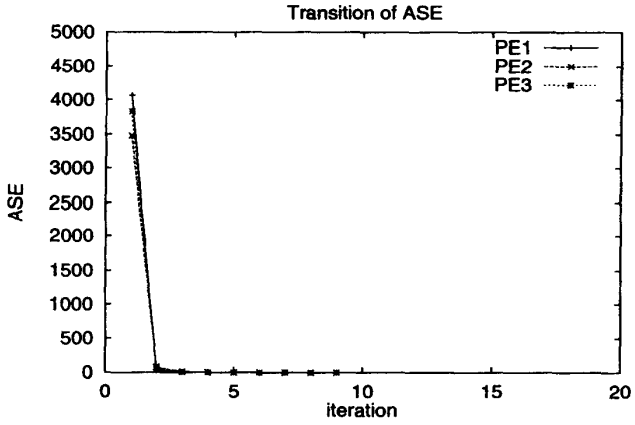

図3 ASE の収束状況

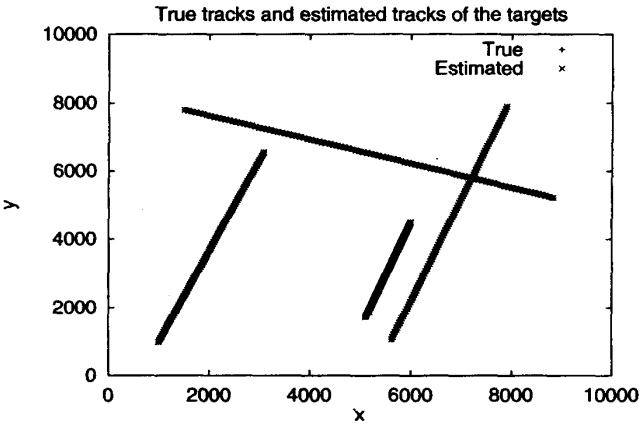

図4 推定軌跡

6.2 収束条件と実行時間の関係

収束条件の変化に伴い，時間も当然変化する。条件がきついと収束に時間がかかるが，解の精度は良くなる。反対に，条件が緩いと収束には時間がかからないが，解の精度は悪くなる。

一般的に収束条件を厳しくすれば，より良い解が得られるが，システムによってはそこまで厳密な解を必要としない場合もある。例えば非常に入力が多く，推定を数多くこなさなければならない場合，一つの推定に時間をとって精度を高めることより，入力に追い付きながら推定をこなすことの方が重要である。

ここで，精度を落す方法について考える。このシステムは ASE によって処理を進めている。具体的には収束条件の判定に使われたり，最良な推定初期状態を選ぶ時などにも使われる。つまり，このシステムにおいての精度は ASE の値に依存している。従って，収束条件である（pre__ ASE-ASE）の値を緩和することによって，精度を落すことが出来，処理の高速化が図れる。ここで，pre__ASE とは，前 iteration での ASE の値である。その図5を示す。

図5 収束条件と実行時間

pre__ASE-ASE	iteration	final ASE	time
0.00001	10	0	21.25
0.0001	10	0	21.00
0.001	10	0	21.00
0.01	10	0	21.00
0.1	9	0.000127	21.00
1.0	7	0.268100	19.00
10	6	0.742492	17.00
100	4	6.588673	11.50
1000	4	6.588673	11.50
10000	2	30.289436	8.00

前論文では，（*pre__ASE-ASE*）$<10^{-5}$であったが，今回100にまで緩和しても，ASE は6.6程度に収まることが分かる。推測すべきデータのオーダーが1000程度であることを考えても十分小さいといえる。逆に実行時間は 4 / 7 程度にまで短縮される。

6.3 曲線推定の実験

次に実時間処理を実装したシステムに関する実験を行なった。今回は一度の推定に使われるデータ数は100とし，次の推定でずらすデータの個数は10としている。このことは，一度の推定中に新たに入力されるデータの個数が10個であるような環境を想定しているということである。また，今回は推定回数を90回に固定した。従って，全部で使われるデータ数は，100＋10×89＝990ということになる。また，この推定では精度は0.01にした。

今回使用した曲線の軌跡は図6，7であり，推定結果は図8，10のようになる。入力軌跡1に対しての推定結果は図8のようになる。この時の各推定での

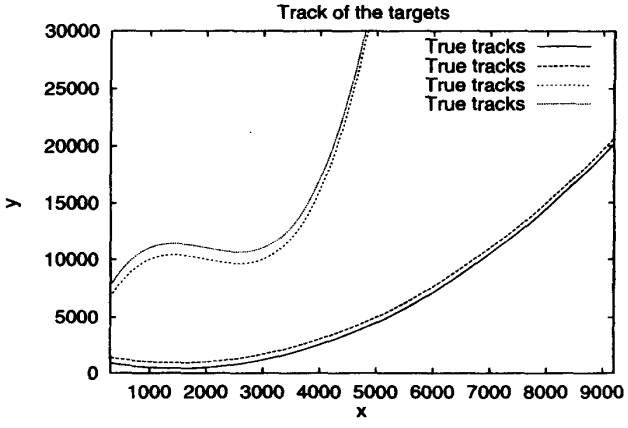

図6 入力軌跡1

iteration の数は図9のようになる。また，入力軌跡2に対しての推定結果は図10のようになる。この時の各推定での iteration の数は図11のようになる。これらの結果を見ると，入力軌跡1のような緩やかな曲線に対しては，かなりの精度で推定可能であることが分かった。しかし，入力軌跡2の3次曲線のように曲率がきつくなると少し誤差が出ている。各 iteration の

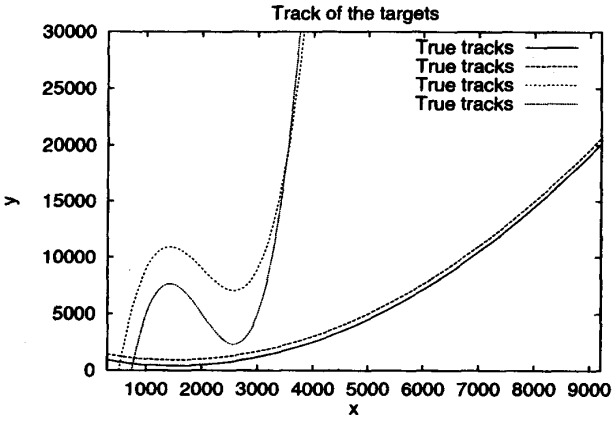

図7 入力軌跡2

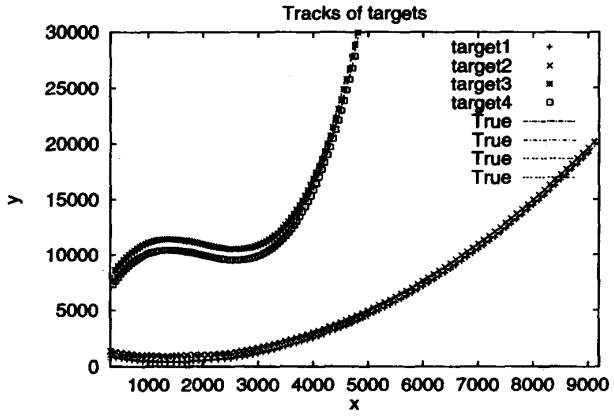

図8 入力軌跡1に対する推定軌跡

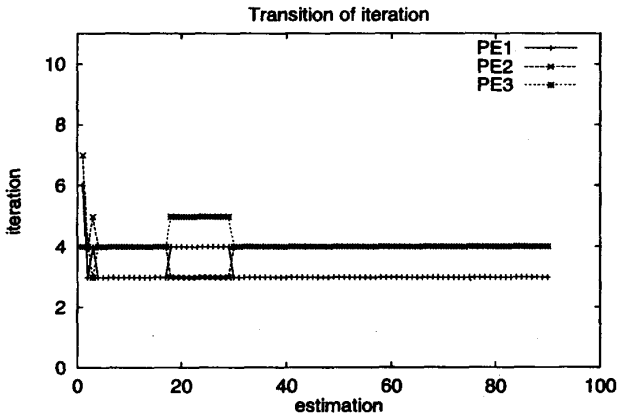

図9 iteration の推移

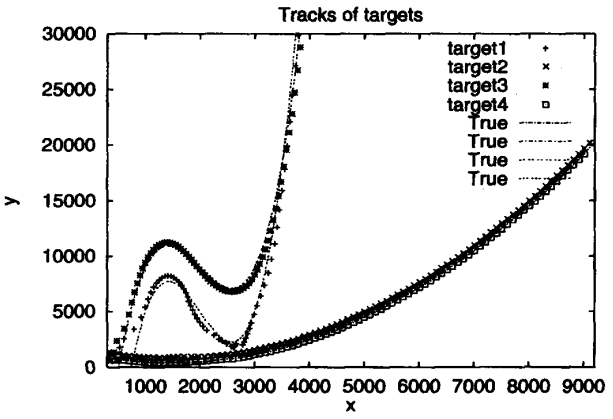

図10 入力軌跡2に対する推定軌跡

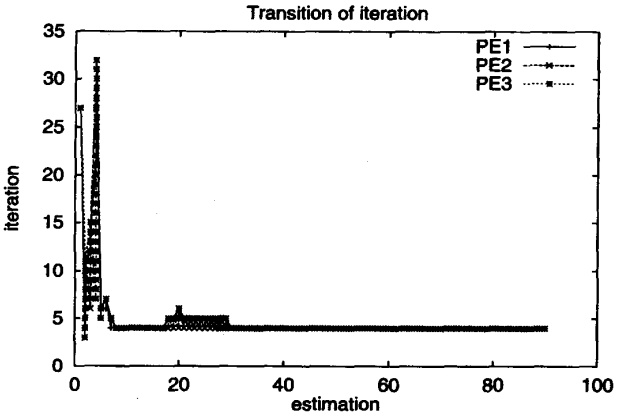

図11 iteration の推移2

推移を見ても図11の方が iteration の数が多くなっている。また，曲率が大きくなっている時に iteration の数が増えている。これは収束に手間を要したことを示している。

7．おわりに

多体軌跡推定の問題は従来の方法では1つの問題解決 PE で集中的に処理されていた。またこの問題の解空間には無数の局所解が存在するため局所解に陥ってしまうことが多く，最適解が得にくいという欠点があった。それを改善するために確率的探索である焼鈍法が導入されたが，これは処理速度が大幅に遅くなってしまった。

一般に分散処理を行ない協調することの利点として，解の多様性が増し局所解に陥ることが減少すること，処理能力が向上することなどが挙げられる。

そして，通信戦略としてローカル法を採用し，従来の手法を元に複数の問題解決 PE を存在させ，分散協調処理を行なう手法がシミュレーションされ，効果を得た。この結果は多体軌跡推定の問題において分散協調処理が非常に有効であることを示すものである。

そこでその分散協調処理を実際に分散化された環境に実装し，性能と速度をシミュレーションと実験比較した。その結果として性能を落すことなく実装でき，その速度はシミュレーションの2.44倍であることが分かった。これによって実際の分散環境でもこの手法が有効であることを改めて証明した。

また，より現実的なシステムを目指し，実時間処理システムへの改良を行なった。その中で，より高速な処理を実現するために，収束条件を緩和することによって推定精度を落し，実行時間とのバランスをとることを試みた。さらに曲線を細分化し直線近似することにより推定することに成功した。これに誤差を加えても推定できることも分かった。

最後に一度の推定で使用する入力数を変化させて実験を行なった。ここでは，緩い曲線に対しては入力数を減少させることにより処理を高速化できた。また，曲率の強い曲線においても微妙にバランスを整えることによって推定可能であることが分かった。

今後の課題としては，実際にセンサを使って，システムにデータを入力させ推定を試みることが挙げられる。

参考文献

1）Pei-yih Ting and Ronald A. Iltis, Multi Target Motion Analysis in a DSN, IEEE Transactions on System, Man, and Cybernetics, vol. 21, no. 5, pp. 1125-1139, Sep/Oct. 1991.

2）D. N. Jayasimaha, S. S. Iyengar and R. L. Kashyap, Information Integration and Synchronization in a Distributed Sensor Networks, IEEE Transactions on System, Man, and Cybernetics, vol. 21, no. 5, pp. 1032-1043, Sep/Oct. 1991.

3）Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill, Cooperation Through Communication in a Distributed Problem Solving Network, Michae N. Huhns ed, Distributed Artificial Intelligence (1985), Morgan Kaufmann Pub, pp. 29-58

4）三谷章男，分散協調による多体軌跡推定，九州大学修士論文（1996）