マイクロコンピュータ 47－3
（１９８７．１２．１）

衝突回避へのトランスピュータの応用

元　村　直　行
（株）安川電機製作所　　研究所

実時間での２腕の衝突回避問題を解くための分散システムをoccamとトランスピュータを用いて構築した。　システムはモジュール化されていて、各モジュールはローカルメモリとそれに対する処理プロセスを持ち、お互いの通信と同期はメッセージパッシングによっている。

メッセージパッシングによる通信と同期はoccamの得意とするところであるが、各モジュールの動作時間帯や速度が異なる場合にデッドロックを回避するには工夫が必要であり、現実的な方法としては、dummy情報を積極的に活用して通信の順序を確立するのが有効であることを確認した。

THE TRANSPUTER SYSTEM FOR COLLISION FREE OPERATION OF TWO ROBOTS

Naoyuki　Motomura
Research Laboratory,　YASKAWA Electric Mfg. Co., Ltd
2346, Fujita, Yahatanishiku, Kitakyushu-shi, 806 JAPAN

The distributed system using occam and transputer for real time collision-free operation of two robots has been constructed.　The system is divided in several modules with local memory and local process.　The modules communicate and synchronize each other by message-passing.

The message-passing is easily realized using occam, but a dummy information is needed to avoid a deadlock when modules works in different time and speed.　It is confirmed useful to fix the sequence of communication among modules using a dummy information in such case.

１．はじめに

マイクロコンピュータは、ロボットコントローラの情報処理においても中心的な役割を果たしているが、ロボットに要求される機能の増大に伴い、その処理すべき情報量も増加している。　特に、周囲の状況を的確に判断して適切な行動をするような、自律型のロボットコントローラにおいては、膨大な量の情報を処理しなければならない。　処理能力を向上させるには、高性能のマイクロコンピュータを使用するか、処理機能を複数のマイクロコンピュータにうまく分散させて並列処理を行なうことになるが、我々は、特に後者に興味があり、実時間での２腕の衝突回避問題を例題として分散型システムの研究を行なっている。　本稿ではこの２腕協調システムのモジュール構成、通信と同期、割付などについて述べるが、本システムはoccam(1)で記述され、５個のトランスピュータ(2)上で実行されている。

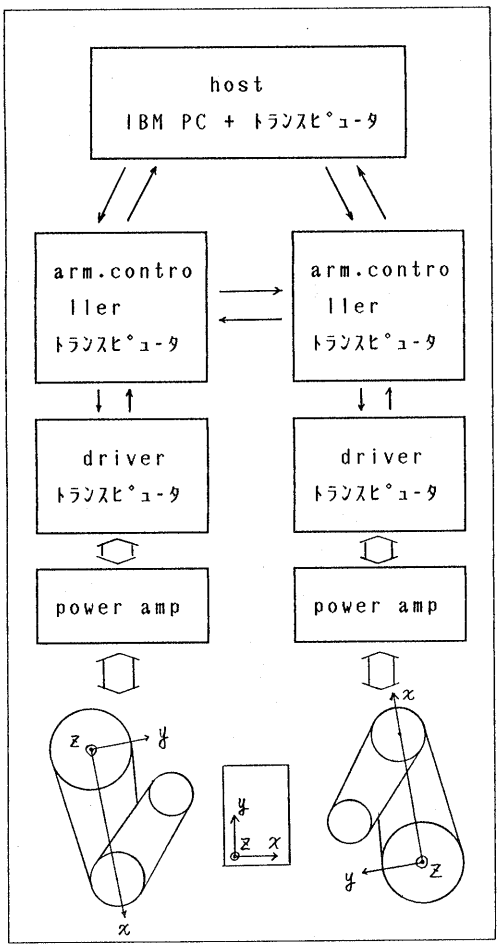

図１　ハードウェアの構成

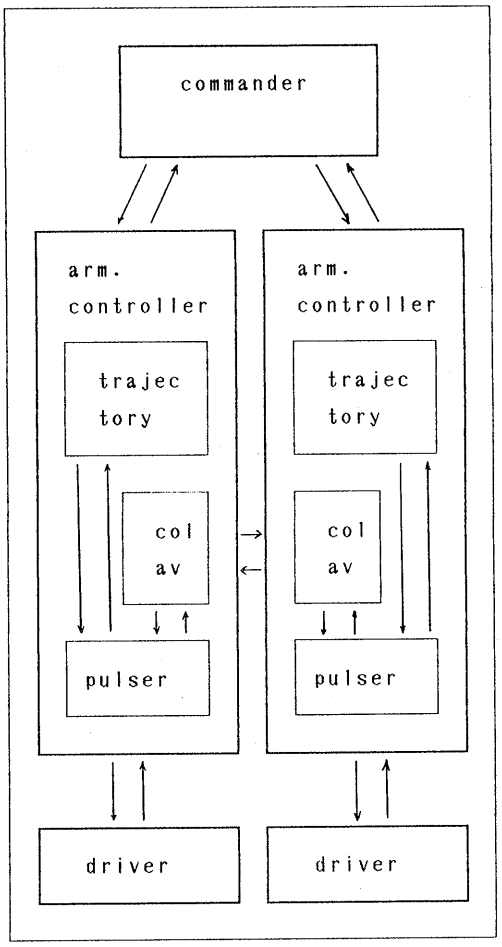

図２　ソフトウェアの構成

２．ハードウェアの構成と問題の設定

最初に２腕協調システムのハードウェア構成について説明するが、図１に示すように、２つの腕が、お互いに衝突の危険性のある位置に置かれていて、その間には、共通の作業テーブルがある。　各腕は、お互いに通信しあう個別のコントローラにより制御される。

- 腕　　２リンクと手首を持ち自由度４
  パルスモータ駆動
  腕自身の座標系
- arm.controller　２個のトランスピュータ
  相手とホストとの通信機能
- 作業テーブル　作業座表系
- host　ＩＢＭ ＰＣ ＋トランスピュータボード(3)

さて問題は、２台のarm.controllerの協調動作により、相手との衝突について考慮されていない動作指令を実行することである。

- 相手との通信　関節座標（作業座標系）や動作状態
- 問題　相手との衝突を避けながらの動作指令の実行

３．ソフトウェアの構成

前の項で述べた問題を解くためのソフトウェアの構成について述べるが、全体の構成は図２に示すように、commander, arm.controller, driverの３つのモジュールからなり、arm.controller, driverは両腕用に同じモジュールが２つずつ起動される。　各モジュールはローカルな情報記憶部（ローカルメモリ）とそれに対する処理プロセスを持ち、モジュール間はチャネル（図２の矢印）を介してのメッセージパッシングにより結合される。　モジュール間での情報の共有（共有メモリ）は、アクセスの競合がなくて、それらが別々のトランスピュータ上で実行されることがないときに限って許している。

３．１　commander

本システムとユーザとのインタフェースの役をしており、作業座表系で記述されたコマンドをarm.controllerへ送信したり、システムの状態を表示したりする。　コマンドは次のような３群に分類される。

- システム設定　作業座表系と腕座表系の変換マトリクスの作成や、
  原点復帰など
- 両腕のモニタ　各関節角や、作業座表系や腕座表系での手首の位置や
  姿勢の表示など
- 両腕の動作　ジョギングやリンク動作や直線動作など

３．２　arm.controller

このモジュールはtrajectory, pulser, colavの３つのサブモジュールで構成されており、commanderから送信されてきた作業座表系でのコマンドを、腕自身の座標系で解釈し、相手と通信して衝突を回避しつつ実行する。

３．２．１　trajectory

本モジュールの主要な仕事は動作指令を実行することである。　目標点は作業座表系で送られてくるので、それを腕座表系に変換しリンク動作や直線動作などの方式により各関節の回転角を計算する。　なお、この計算の段階では相手との衝突については何も考慮しない。　pulserへ送信されるメッセージは

arm.state　　動作状態が変化するとき送信
　　arm.state := waiting | jogging | moving | working
pulse.ready　払い出しパルスの準備ができたとき送信
dummy　　自分を無視して進んでほしいとき送信

traje cola	pulse.ready	arm.state	dummy
avoid	SEQ to.driver ! pulse.fr.colav renew.position() PAR to.colav ! position to.trajec tory ! nack	SEQ to.driver ! pulse.fr.colav renew.position() PAR to.colav ! position.arm .state to.trajec tory ! ack	SEQ to.driver ! pulse.fr.colav renew.position() PAR to.colav ! position to.trajec tory ! ack
wait	PAR to.colav ! dummy to.trajectory ! nack	PAR to.colav ! arm.state to.trajectory ! ack	PAR to.colav ! dummy to.trajectory ! ack
dummy	SEQ to.driver ! pulse.fr.traje renew.position() PAR to.colav ! position to.trajec tory ! ack	PAR to.colav ! arm.state to.trajectory ! ack	PAR to.colav ! dummy to.trajectory ! ack

図３　pulserの動作

であり、管理情報は

・手首や関節の位置
・各関節角
・座標変換マトリクス
・目標点

などである。

３．２．２　pulser

本モジュールは、trajectory、colavからのメッセージの組合せによってどのような動作をするか図３に示す。　例えば、colavからavoid，trajectoryからpulse.readyを受信したとき、まずdriverへpulse.fr.colav(colavで計算された退避のための払い出しパルス量）を送信し、各関節角と手首や関節の位置の更新を行ない、手首と関節の位置（作業座表系）をcolavへ送信する。　図中、SEQ，PARはそれぞれシーケンシャル、及び、コンカレントな処理を意味し、!，？は送信、受信を意味する。　なお、このモジュールはtrajectoryと情報を共有するが、両者の動作はシーケンシャルなので問題はない。

３．２．３　colav

本モジュールは、pulser及び相手のcolavからの情報により図４に示すような動作をする。　この図から分かるように、少なくとも一方の腕の位置が変化したときに、サブルーチンeva.joint.vector()で相手との衝突の危険性がチェックされ、安全なときはpulserへdummyが送信される。　危険なときにはまず、動作状態などから優先権の有無を判断し、自分に優先権があるときは、危険度に応じてそのまま動作を続けるか（pulserへdummyを送信する）、安全になるまで待つ（pulserへwaitを送信する）ようにする。　逆の場合には、相手との衝突を回避するように各関節の回転角を計算してpulserへ送信する。　管理情報は

・両腕の手首と関節の位置、目標点

partner pulser	position. arm.state	position	arm.state	dummy
position. arm.state position arm.state dummy	SEQ eva.joint.vector() IF safe to.pulser ! dummy NOT safe avoid ! wait ! dummy		to.pulser ! dummy	

図４　colavの動作

3.3 driver

各関節のパルスモータは2相の正弦波状の信号で駆動されるが、本モジュールは、pulserからのパルス払い出し指令の分だけ、この駆動信号の位相を変化させる。

4. arm.controllerのサブモジュール間の通信と同期

occamでは同期通信をサポートしているので、通信と同期の問題を一緒に処理できるので便利であるが、本サブモジュール間の通信のように動作時間帯が必ずしも一致しない場合には少し工夫が必要である。まず最初に、各サブモジュール

	phase0	phase1	phase2
trajectory		to.pulser ! arm.state | pulse.ready | dummy	fr.pulser ? ack | nack
pulser	to.colav ! position	PAR fr.trajectory ? arm.state | pulse.ready | dummy fr.colav ? avoid | wait | dummy	PAR to.trajectory ! ack | nack to.colav ! position | arm.state | position. arm.state | dummy
colav	SEQ fr.pulser ? position PAR to.partner ! position fr.partner ? position	to.pulser ! avoid | wait | dummy	SEQ fr.pulser ? position | arm.state | position. arm.state | dummy PAR to.partner ! 同上 fr.partner ? 同上

図5　trajectory, pulser, colav間の通信

の動作時間について考えてみるに、trajectoryはcommanderからの指令がないときは休んでいるが、colavはたとえ自腕のtrajectoryが休んでいても、相手の腕が動いている限り休むわけにはいかない。　pulserは、この両者と通信するわけだが、この様な場合にデッドロックに陥ることなく通信を継続するには、dummy情報を”先に進め”といういうように前向きに解釈して、ある一定の通信の順序パターンを形成するのが良いようである。　なお、このようにしても時間的な制約は発生しないこと、すなわち本例でいえばcolavの処理時間が長くなったとしても他のモジュールが待つだけであることに注意する。　arm.controllerのサブモジュー

```
PROC arm.controller( VAL INT arm.id,
     CHAN OF ANY fr.driver, to.driver, fr.partner, to.partner )
  ...
PROC driver( CHAN OF ANY fr.ac, to.ac )
  ...
CHAN OF ANY ac1.to.ac2, ac2.to.ac1 :
CHAN OF ANY ac1.to.driver1, driver1.to.ac1 :
CHAN OF ANY ac2.to.driver2, driver2.to.ac2 :
PLACED PAR
 PROCESSOR 0 T4
   PLACE ac1.to.ac2 AT 2 :
   PLACE ac2.to.ac1 AT 6 :
   PLACE ac1.to.driver1 AT 1 :
   PLACE driver1.to.ac1 AT 5 :
   arm.controller( 0, driver1.to.ac1, ac1.to.driver1,
                      ac2.to.ac1, ac1.to.ac2 )
 PROCESSOR 1 T4
   PLACE ac1.to.ac2 AT 6 :
   PLACE ac2.to.ac1 AT 2 :
   PLACE ac2.to.driver2 AT 1 :
   PLACE driver2.to.ac2 AT 5 :
   arm.controller( 1, driver2.to.ac2, ac2.to.driver2,
                      ac1.to.ac2, ac2.to.ac1 )
 PROCESSOR 2 T4
   PLACE ac1.to.driver1 AT 4 :
   PLACE driver1.to.ac1 AT 0 :
   driver( ac1.to.driver1, driver1.to.ac1 )
 PROCESSOR 3 T4
   PLACE ac2.to.driver2 AT 4 :
   PLACE driver2.to.ac2 AT 0 :
   driver( ac2.to.driver2, driver2.to.ac2 )
```

図６　コンフィギュレーション

ル間の通信のパターンを図５に示すが、phase0はシステム起動時に一度だけ実行されるもので、それ以後は、phase1とphase2とが繰り返される。

５．マルチトランスピュータへの割付

現在のところ、本システムは５個のトランスピュータ上で走っているが、モジュールcommanderはホスト上のトランスピュータ上で走り、arm.controller, driverはそれに接続されているトランスピュータネットワーク上で走る。arm.controller, driverをトランスピュータネットワークにダウンロードするには、図６に示すようなコンフィギュレーションのためのプログラムを書く。

図６の第６行は、２つのarm.controllerの間の通信チャネルの宣言であり、第１２、１３行でそれらをトランスピュータ０のリンク２と６に割り当てている。また、第８行はarm.controller1とdriver1との間の通信チャネルの宣言であり、第１４、１５行でそれらをトランスピュータ０のリンク１と５に割り当てている。そして、第１６行で、これらのチャネルを引き数としてarm.controllerをコールしている。なお、腕のid番号も引き数として渡しているのは、原点復帰時の関節角と作業座表系、腕座表系の変換マトリクスを作成するための数値が両腕で異なるためである。

６．おわりに

本システムが、実時間での２腕の衝突回避問題を解くのに有効であることはすでに確認できているが、衝突回避戦略はまだ未熟であり大いに精進しなければならない。また、この戦略が高度化するにつれてcolavの処理すべき情報量が増大するのは明かであり、いずれこのモジュールはarm.controllerから独立して別のトランスピュータ上で走ることになると思われる。

参考資料

(1) occamプログラミングマニュアル　啓学出版
(2) Transputer Reference manual　INMOS
(3) IMS B004 evaluation board