計測自動制御学会東北支部 第 260 回研究集会 (2010.10.29)
資料番号 260-7

ペトリネットを利用した搬送系の構築

Construction of Transfer System Using Petri Net

○近藤優丞*, 有我祐一*, 及川雄大*, 渡部慶二*, 遠藤茂*
○ Yusuke Kondo*, Yuichi Ariga*, Takehiro Oikawa*, Keiji Watanabe*, Shigeru Endo*

*山形大学

*Yamagata University

キーワード： ペトリネット (Petri net), 自動搬送装置 (Automated Guided Vehicle), LEGO Mindstorms

連絡先： 〒992-0037 米沢市城南 4-3-16 山形大学 工学部 応用生命システム工学科 有我研究室 近藤優丞, E-mail: yusu_k_on_1341@yahoo.co.jp

1 はじめに

我々の身の回りには実に多くの組込み機器があふれている。携帯電話やデジタルカメラやカーナビゲーションのような身近で普段よく使うものがまず思い浮かんでくるが、それだけではなく鉄道や工場のロボット、医療機器や自動車、あるいは人工衛星など世界中いたるところに組込み機器は存在している。組込み機器はそれぞれが特定の用途に特化していて、同じ「組込み機器」というくくりの中でもほかの組込み機器とは全く違う形をしており、その形は多種多様である。

これら組込み機器では、パソコンのようにソフトウェアの不具合に対してインターネット経由などの修正パッチを適用して対応するということが難しい。一度不具合が発生すれば即座に故障とみなされ、開発メーカが自主回収や無料修理を行うことになるだろう。もしも医療機器や交通にかかわる組込み機器であれば、人の命にかかわる重大な問題となりえる。このようなことを回避するためにも組込み機器では不具合を排除した高い信頼性が求められている。

不具合を排除するための方法として「プログラムを実機に搭載する前に入念なシミュレーションを行う」ことがあげられる。そこでシミュレート能力に優れるペトリネットでシステムをモデル化することで、入念なシミュレーションというものが可能になるのではないかと考えられる。これはフローチャートなどの静的なモデル図では実機の動作を追うときに分かりにくいが、ペトリネットであればトークンの移動によって視覚的に分かりやすいようにシステムを動的なモデルとして再現できるという特性があるからである。

本研究では自動搬送装置 (Automated Guided Vehicle：AGV) を取り入れた搬送システムを作り、AGV のプログラムをペトリネットでモデル化している。AGV プログラムのモデル化を通してペトリネットのモデル化能力の検証をするとともに、モデル化に際する問題点を明らかにしたい。検証のためのモデルとして、LEGO 社製マインドストームを AGV として利用した搬送システムを作成している。

2 ペトリネット

2.1 ペトリネット

2.1.1 概要

ペトリネットは並行的・非同期的・非決定的なシステムを表すための数学的モデルであり、離散事象システムのモデル化に広く用いられる。

2.1.2 ペトリネット

ペトリネットは次の 4 つの要素から構成される。

① プレース：状態 ○
② トランジション：事象 |
③ トークン：要素 ●
④ アーク：要素の流れ →

トークンの移動によってプレース配置の組み合わせが変化し、ペトリネットは状

態の遷移をグラフィックモデルとして表すことが出来る。

3　AGV を適用した搬送モデルの設計と制御

3.1　搬送モデルの設計と制御方法

AGV を適用した搬送モデルを構築するに当たり、搬送モデル全体の構成の設計をし、それに合わせて AGV の基本設計を行い制御方法を決定する必要がある。

本研究では DSP コンピュータで制御される 2 台の台車と、2 台の AGV によって構成される搬送システムをモデルとして構築した。AGV モデルとして、LEGO 社のロボット製作キットであるマインドストーム NXT を用いている。このマインドストームのパーツを使って AGV を組み立て、専用プログラミング言語である「NXT ソフトウェア」によって制御プログラムを作成する。

AGV を走行させる搬送ルートは 2 つ。2 台の DSP 台車と搬送先の荷台を結ぶ直線上に配置したレールの上。そして荷台と DSP 台車を結ぶように配置した誘導ルート上である。誘導ルートには、市販のビニールテープを使用し、経路選択の制御には光センサを用いる。

3.2 搬送モデル

搬送モデルは 2 台の AGV、2 台の台車、荷物のロード/アンロードポイントになる荷台、ライントレース用誘導ルートからなる。台車、AGV がそれぞれ決められたルートで荷物を搬送することによって、荷物がシステム内を循環するようになっている。搬送モデルの概略図と実際に作成したモデルを図 1 (a),(b)として示す。

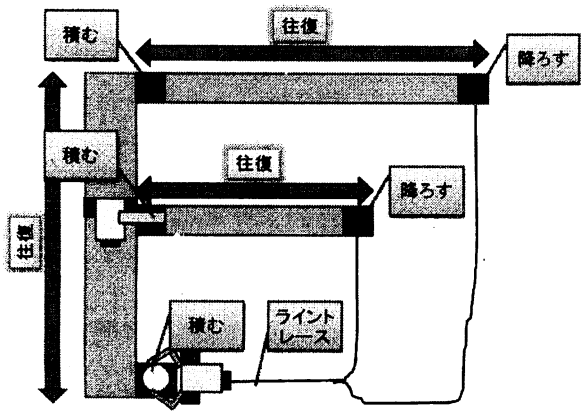

図 1 （a）搬送モデルの概略図

荷物は白と黒の 2 種類あり、ライントレースをする AGV は白い荷物を荷台から遠い台車へ、黒い荷物を荷台から近い台車へと搬送する。台車は PSD センサで荷物の有無を判別し、荷物がある場合は図 1(a)で左の方へ移動する。荷物を左へ運んだ台車はそのまま待機し、左側にある AGV によって荷物が運び出されるまで待機し続ける。

ライントレースのルートは黒いビニールテープで作ってあり、トレースを終了して停止する場所には黄色いテープを張り付けて光センサへ返す値を調整している。

3.3　AGV モデルの構造

本研究では 2 台の役割の違う AGV を用意している。本体は共に LEGO 社製マインドストーム NXT であり、これには A, B, C という 3 つの出力ポートと 1，2，3，4 という 4 つの入力ポートが用意されている。それぞれの AGV を NXT 1、NXT 2 と呼びおのおのについて説明する。

<NXT1>

2 つの台車と荷台を結ぶレール上を移動する。ポート 1 に接続した光センサを使ってアームの範囲内に荷物があるかどうか調べる。光センサが閾値より大きい値を感知したら台車から荷物を取り上げ荷台へと搬送する。荷台まで移動した後は光センサを利用し荷台が空き状態なのかを判断する。空き状態でなければ空くまで待機し、空き状態であれば荷物を置き台車へ移動する。

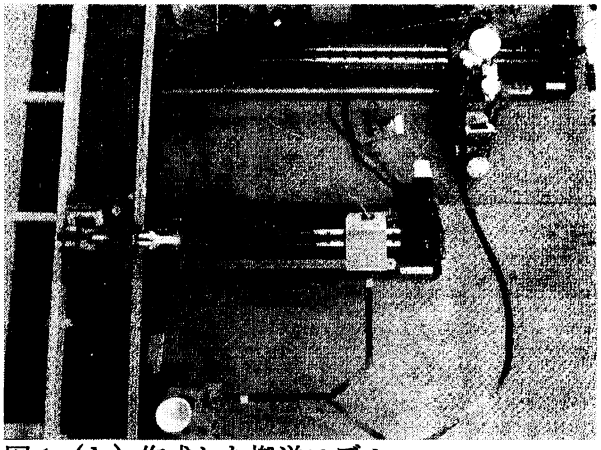

図 1 （b）作成した搬送モデル

＜NXT2＞

ロードした荷物を誘導ルートを通って DSP 台車へと運ぶ。初期状態ではポート３に取り付けた超音波センサが荷物を感知するまで待機し続ける。ロードしたらポート１の光センサで荷物の反射光の値を読み、閾値より大きければ台車１へ、小さければ台車２へ誘導するルートに乗り荷物を搬送する。ライントレースはポート２に繋がる光センサで誘導ルートの右エッジを読み取って行う。台車に到着しアンロードしたら反転して辿ってきた誘導ルートを使って再び荷台まで戻る。

3.3 制御プログラムの構造

制御プログラムは「NXT ソフトウェア」がインストールされたパソコンを使って作成する。作成したプログラムは USB ケーブルを使って直接マインドストームへと接続して転送する。二台の AGV の制御するためのプログラムの詳細について以下記述する。

＜プログラムの流れ　NXT1＞

[Step1]

台車 2 で荷物があるかどうか光センサで確認。センサが閾値より大きい値を感知したらロードして荷台へ搬送。

[Step2]

荷台へ移動したら、光センサで荷台が空き状態なのか調べる。光センサが閾値より大きい値を感知したら空いていないので待機。空いたら荷物をアンロード。

[Step3]

台車２へ移動し、荷物があるかどうか光センサで調べる。光センサが閾値より大きい値を感知したらロードして荷台へ移動し[Step2]へ。小さい値を感知した場合は５秒間待機し、それでも光センサが荷物を感知しなければ台車１へ移動。

[Step4]

光センサで荷物があるかどうかを調べる。光センサが閾値より大きい値を感知したら荷物があるのでロード。荷台へ搬送し[Step2]へ。なければ荷物を感知するまで待機。

＜プログラムの流れ　NXT2＞

[Step1]

超音波センサで荷台に荷物があるか確認。感知しなければ待機。感知したらロード。

[Step2]

ロードした荷物を光センサで確認。閾値より大きな値を感知したら反転して台車１への誘導ルートに乗る。小さな値を感知したら反転して台車２への誘導ルートに乗る。

[Step3]

光センサで誘導ルートとなるビニールテープの右エッジを感知して台車までライントレースを続ける。

[Step4]

光センサが閾値より大きい値を感知したら停止。アンロード。

[Step5]

反転し、光センサで誘導ルートとなるビニールテープの左エッジを感知して荷台までライントレース。

[Step5]

光センサが閾値より大きい値を感知したら停止。[Step1]へ戻る。

4　NXT プログラムのモデル化

当初は前項のように Step ごとに分割したものをそれぞれモデル化して組み合わせることで、楽にプログラムのモデル化をすることが可能であると考えていた。しかしそれではシステムを再現した正しいモデルにならないことが分かった。以下、プログラムの内容が比較的簡単な NXT1 のプログラムをモデル化することを通して正しいモデルにならない理由、解決方法について説明していく。

本研究ではモデル化をするためのエディタとして、インターネット上でフリーソフトとして公開されている「Platform Independent Petri net Editor2 (PIPE2)」というソフトを使用している。

4.1 Step 分けしたプログラムのモデル化

Step 分けした通りに順番にプログラムをペトリネットで表現していき、それを１つにまとめると図２のようになる。これは初期マーキングが復元できるペトリネットなので、往復しながら荷物を搬送し続けるという NXT1 の仕事を見掛け上正しくモデル化しているとはいえる。ペトリネットとしても安全に動作する。

しかし図２のネット図では、プログラムを実行したときに実機が見せる動作とまったく違う構造になっている部分がある。それが図２において破線で囲んである「光センサ」が関わる部分である。

容量が１で２出力をもつプレースでは、トークンを持ったときに発火させることができるトランジションは１つだけであり、どちら

を発火させるかはランダムになる。図２の光センサの状況も同様で、いまのままでは光センサの応答が完全にランダムということになる。だが実機ではランダム要素はなく光センサで荷物の有無を確認するのだから、そのように正しくネット図を修正しなくてはならない。

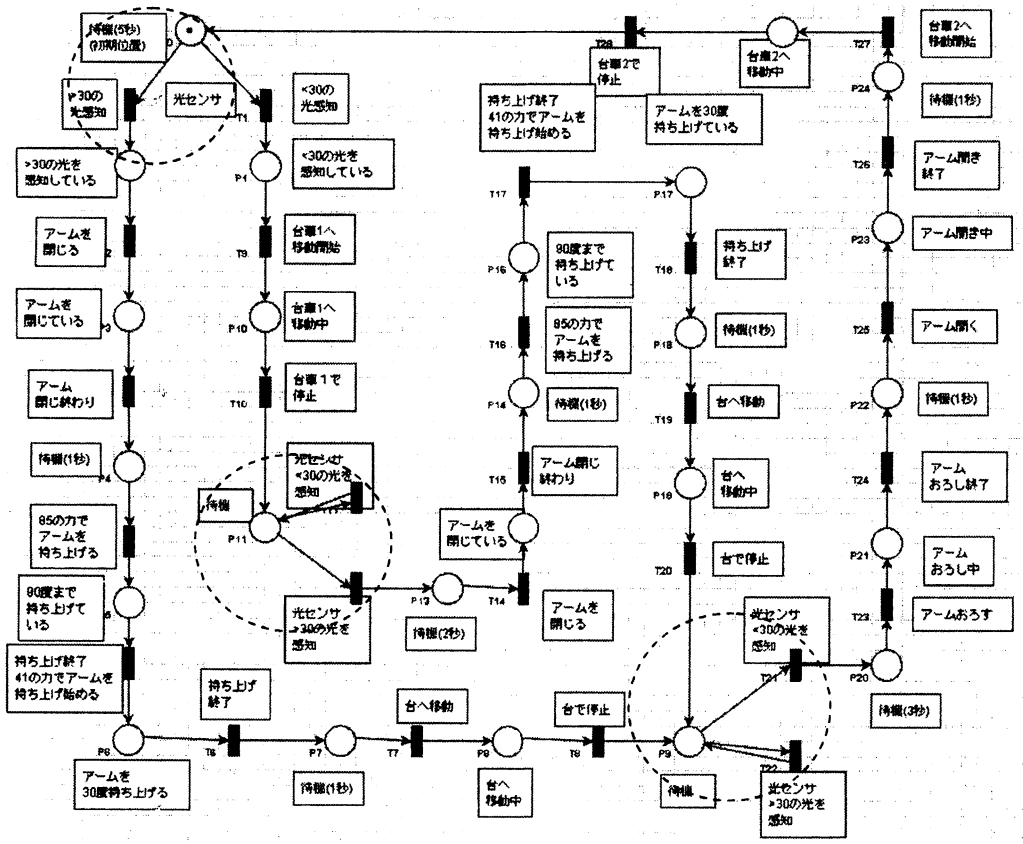

図２ NXT1 のプログラムに沿ってモデル化したペトリネット

4.2 正しい動作をするモデルへの修正

このようにおかしな部分が出てくるのはなぜなのか。それはプログラムだけでモデル化しようとすると、ペトリネットとして情報が足りないという状態が起きるからである。そこで不足している情報を補ってネット図を修正することにする。

前述したように光センサが関わる動作の再現が不完全なので、ランダム性をなくして「光センサが荷物の有無を判断している」という現実の動作を再現できるようにネット図へ描きかえることを考える。ここであらためて光センサを搭載している理由を考えると「荷物の有無の判断」というのがそれになる。現在の図２の状態では光センサが何にも依存せずに荷物の有無を判断しているというネット表現が問題であるから、「荷物の現在位置」と「荷物の色」という情報を追加することで問題の解消を試みる。

荷物の現在位置をあらわすには搬送に関わるNXT２、台車１、台車２の情報を無視することはできないので、これらのこともネット図へと加えることになる。そうすることでより正確なモデル化が行えると考えられる。

そうしてこれらの情報を追加して拡大したのが図３である。破線で囲まれている部分が荷物についての情報を増やしてランダム性を排除した構造に作り替えた光センサの部分である。点線で囲まれているのが荷物の現在位置を表現するために追加した NXT2、台車１、台車２の部分である。

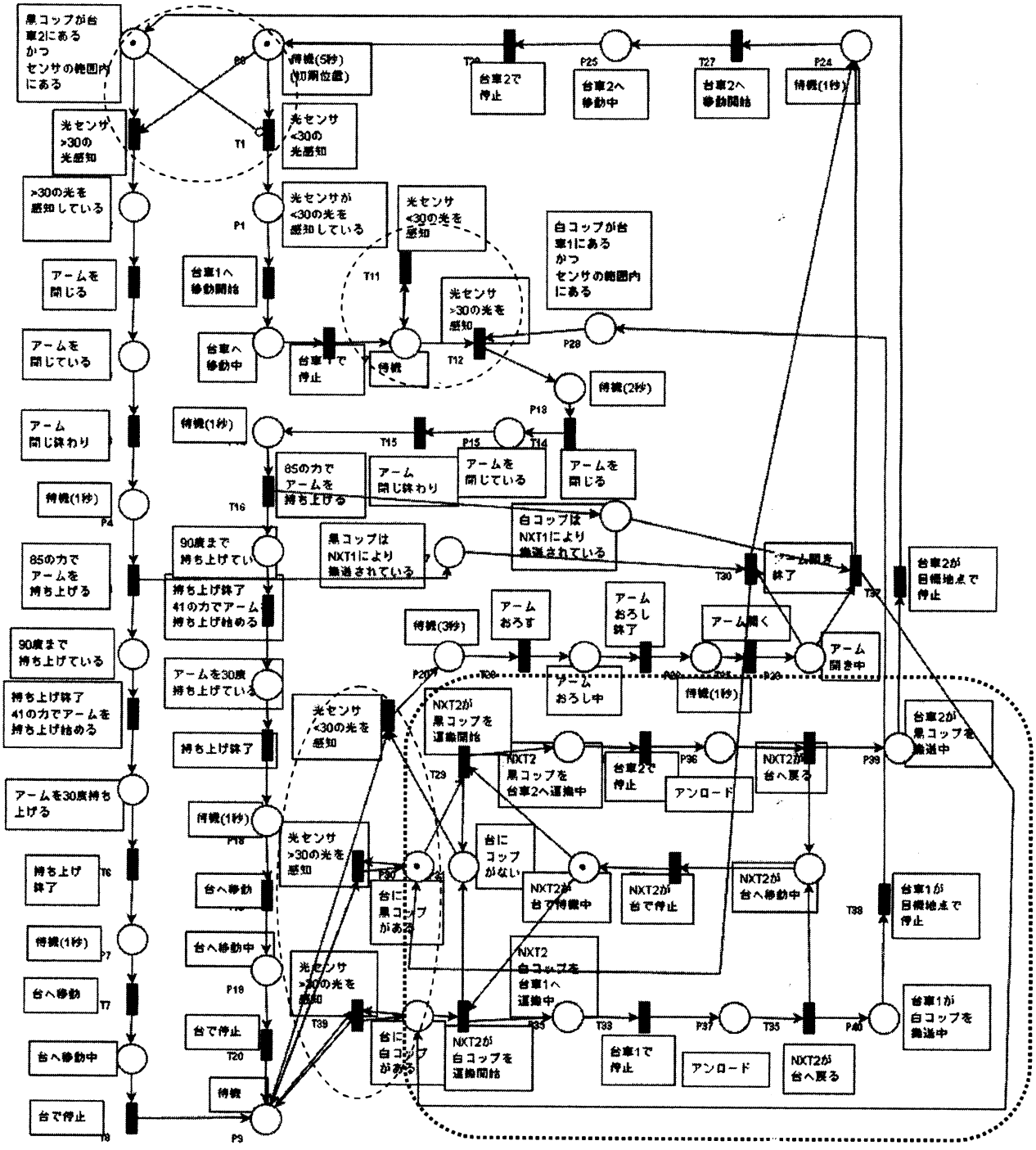

図３　荷物、NXT2、台車１、台車２の情報を追加したペトリネット

5　まとめと考察

本研究ではペトリネットを用いてAGVの制御プログラムをモデル化することで、そのモデル化能力とモデル化に際しておこりえる問題点を検証した。

当初、制御プログラムをそのままペトリネットに置き換えるようにすればモデル化はすぐにできると考えていたが、実際はそうではなかった。搬送システムを作成して検証してみると、プログラムを単独でモデル化しても正しい動作をするモデルは得られず、搬送対象となる荷物の情報とシステム内の他の搬送マシンの情報が必要となることが分かった。

また今回はモデル化したネット図のシミュレーション能力が低いことがわかった。これはネット図の問題というよりはソフトウェアの機能不足が原因であるといえる。複数のトランジションの同時発火機能や階層化のシステムがないために実際の時間の流れを無視したシミュレート内容になったりモデルが大きくなりすぎて見難くなるなどという問題があった。

ペトリネットには広く認知されたソフトウェアの統一規格のようなものがなく、ペトリネットソフトを必要とする研究者が自分の為に自分に必要な機能を実装したソフトウェアをインターネット上で公開している。そのためにインターネットで得られるソフトウェアにはほしい機能が実装されていないことが多い。ペトリネットの有用性が高まればソフトウェアに必要な機能をまとめて共通の様式を定める動きが出てくるかもしれないが、それまでは自分の為のソフトウェアを自作していくしかないようである。

参考文献

（1）　椎塚久雄　実例ペトリネット　コロナ社

（2）　渡辺政彦, 飯田周作, 石田哲史, 山本修二, 浅利康二: UML 動的モデルによる組込み開発-分析・設計・実装・テスト-, オーム社/開発局