シミュレータ作成の手間を削減可能な二部構成シミュレータ設計手法の提案

川元 卓 [a)]・喜多 義弘 [b)]・片山 徹郎 [c)]

Proposal on a Design Method of a Simulator Divided into Two Parts Which can Reduce the Effort of Generating the Simulator

Taku KAWAMOTO, Yoshihiro KITA, Tetsuro KATAYAMA

Abstract

As one of problems in the development of embedded systems, it takes much time to verify the behavior of the system. To solve this problem, the simulator is often used. However, the developer must develop both of the software and simulator because the simulator is generated in hands. Therefore, it is important to reduced the effort of generating the simulator. This paper aims to reduce the effort of generating the simulator, and proposes a design method of a simulator divided into two parts. This simulator is consisted of two parts: a part of the development environment and a part of the runtime environment. Each part can be executed in parallel, and be reused. Therefore, it is possible to reduce the effort of generating the simulator by reusing the each part.

Keywords: Embedded system, Simulator, Design method

1. はじめに

組込みシステム技術は、現在、日常生活において至る所で使われており、我々の生活には欠かせないものとなっている [1)]。組込みシステム技術は日々進歩してきており、平成 20 年 4 月には、日本標準産業分類コードに「組込みソフトウェア業」が追加され、「組込みソフトウェア業」が 1 つの産業として確立された [2)]。

この組込みシステムの開発における問題点の 1 つとして、「システムの動作確認に手間がかかってしまう」ことが挙げられる。

組込みシステムの場合、ソフトウェアはそれ単体で動作することができない。これは、実際に実機にソフトウェアを組込んで動作確認を行わなければならないからである。この実機を用いた動作確認は手間がかかるため、ソフトウェアの開発にかかる時間は大きくなってしまう。

この問題を解決する既存の方法の 1 つに、シミュレータを利用する方法がある。シミュレータを用いることによって、実機を用いることなくソフトウェアの動作確認を行うことができる。しかし、その際、シミュレータの作成は人手で行うため、開発者はソフトウェアの開発と同時にシミュレータの作成も行う必要がある。そのため、シミュレータ作成にかかる手間を削減することが重要となる。

そこで本研究では、シミュレータ開発における手間の削減を目的とし、二部構成シミュレータ設計手法を提案する。提案する二部構成シミュレータ設計手法は、開発環境部および実行環境部の 2 つの部から構成する。このシミュレータにおいて、2 つの部はそれぞれが並列に動作し、再利用が可能である。そのため、一度作成したそれぞれの部を別のシミュレータの作成時に再利用することによって、シミュレータ作成の手間の削減に繋がると考えられる。

本論文の構成は、以下の通りである。

2 章では、二部構成シミュレータを構成している要素について説明する。3 章では、今回試作する開発環境部と実行環境部について詳細に説明する。4 章では、3 章で作成した開発環境部と実行環境部の組合せを変え、二部構成シミュレータを作成し、シミュレータが実機の動作を再現していることと、組合せを変えても、正しく動作することを検証する。5 章では、提案した手法によって作成した二部構成シミュレータの有用性について考察する。

2. 二部構成シミュレータ

図 1 に、提案する二部構成シミュレータ設計手法の全体図を示す。この章では、開発環境部と実行環境部、仮想メモリマップ、共通ヘッダについて、それぞれ説明する。

a)情報システム工学専攻大学院生
b)情報システム工学専攻研究生
c)情報システム工学科准教授

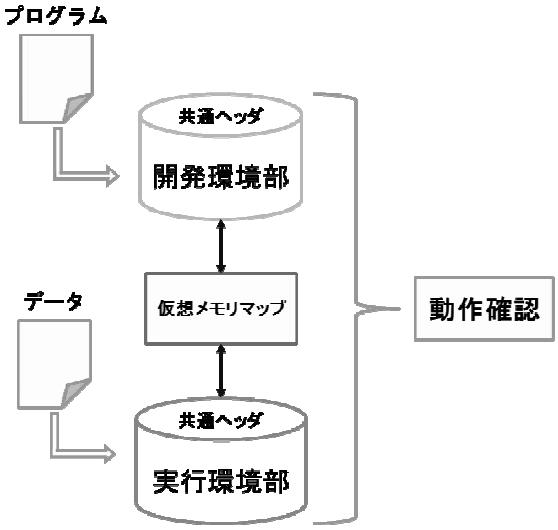

図 1 二部構成シミュレータ設計手法の全体図

2.1. 開発環境部

開発環境部は、主に開発環境に依存する部分をシミュレートするための部である。C や Java といった開発に用いるプログラミング言語や使用するカーネルなど、開発環境に依存する部分をシミュレートする。

この部は、シミュレータ上で再現したいプログラムを入力として与える。入力したプログラムを解析、実行し、得た値を仮想メモリマップの対応するアドレスに書き込む。

2.2. 実行環境部

実行環境部は、開発環境に依存しないシステムの動作をシミュレートする部である。システムの現在の状態を描画したり、センサの値を求めるなど、システムの動作を確認するための部である。

この部は、システムの動作に必要となるデータを入力として与える。入力したデータを元に、仮想メモリマップから値を読み取って、その動作を反映したり、センサなどの値を仮想メモリマップの対応するアドレスに書き込んだりする。

2.3. 仮想メモリマップ

提案する手法は、1 つのシステムを開発環境部と実行環境部に分け、それらの部を並列に動作してシミュレートする手法である。そのため、あるデータが分割した 2 つの部の両方で必要となる時、2 つの部の間でそのデータをやりとりできる仕組みが必要となる。

仮想メモリマップは、開発環境部と実行環境部の間に位置し、2 つの部のデータのやりとりを行う部分である。

データのやりとりをする様子を、図 2 に示す。I/O ポートなどをこの仮想メモリマップに置き換え、開発環境部や実行環境部で求めた値をこの仮想メモリマップの対応するアドレスに書き込むことによって、他方の部からそのデータを読み取ることが可能となる。どのアドレスに何のデータが存在するかという情報は、2.4 節で述べる共通ヘッダに記述する。

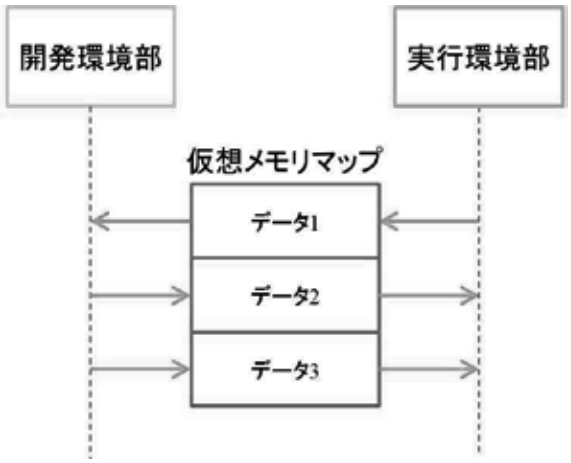

図 2 データのやりとり

2 つの部は、この仮想メモリマップを介してのみデータのやりとりを行う。このため、2 つの部の結合度を低くすることができるので、開発環境部と実行環境部の組合せを変えることによって、別のシミュレータとして使用することが可能となる。

2.4. 共通ヘッダ

提案する手法は、開発環境部と実行環境部を並列に動作してシミュレートする。この 2 つの部の実行周期が異なった場合、仮想メモリマップを介してデータをやりとりするタイミングにずれが生じてしまう。その様子を、図 3 に示す。この図のように、データの書込みと読取りのタイミングがずれてしまうと、シミュレータの精度が悪くなってしまう。そのため、2 つの部の実行周期を揃える必要がある。

また、仮想メモリマップを介してデータをやり取りする時に、どのアドレスを参照するのか、開発環境部と実行環境部で統一しなければならない。この仮想メモリマップの情報を 2 つの部が違う所に記述すると、開発環境部と実行環境部の組合せを変え、別の二部構成シミュレータを作成した場合、その情報を再び統一する必要があり、修正に手間がかかってしまう。そのため、2 つの部に共通する情報を記述する部分が必要となる。

共通ヘッダは、開発環境部と実行環境部の両方に共通して必要となる情報を記述するヘッダである。この共通ヘッダには、実行周期や、仮想メモリマップのどのアドレスに何のデータが存在するのかという情報を記述する。

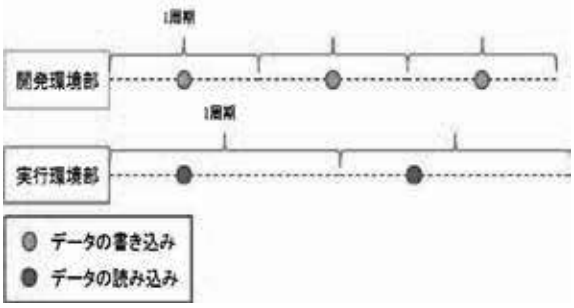

図 3 実行周期が異なる場合

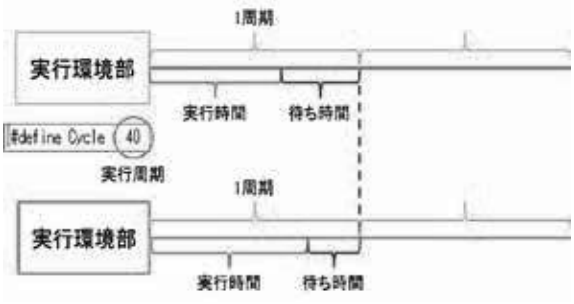

図 4 二部構成シミュレータの実行周期

共通ヘッダに記述した実行周期を用いて、1 周期の実行時間と待ち時間の和が、実行周期となるように待ち時間を計算する。図 4 に、その様子を示す。この図のように、実行周期の時間を指定し、1 周期の実行時間と待ち時間の和が指定した実行周期となるように待ち時間を調整することによって、2 つの部が同じ周期で実行することが可能となる。

また、仮想メモリマップのどのアドレスに何のデータが存在するかについての情報を、共通ヘッダに記述することによって、開発環境部と実行環境部の組合せを変更した場合、この共通ヘッダの部分のみの変更で、別の二部構成シミュレータを作成することが可能となる。

3. 試作したシミュレータ

開発環境部と実行環境部の組合せを変更しても、別のシミュレータとして動作可能であることを検証するために、開発環境部と実行環境部を、それぞれ 2 つずつ試作した。図 5 に、それらのシミュレータを示す。この章では、試作したシミュレータについて説明する。

3.1. 開発環境部のシミュレータ

本論文では、TOPPERS/JSP のバージョン 1.4.3 とバージョン 1.4.4 の 2 つのシミュレータを開発環境部として改変し、使用した。この節では、それらについて説明する。

図 5 作成したシミュレータ

3.1.1. TOPPERS/JSP1.4.3 シミュレータ

TOPPERS/JSP は、μITRON4.0 仕様に準拠したリアルタイムカーネルである [3]。
主な特徴として、読みやすく修正しやすいソースコードや、高い実行性能と小さい RAM 使用量などが挙げられる。

また、TOPPERS/JSP は、Linux および Windows 上で動作可能なシミュレーション環境を提供している。このシミュレーション環境を用いることによって、1 つのプロセスの中で複数のタスクを切り替えて動作することが可能となり、実機に近い開発環境をシミュレートできる。

3.1.2. TOPPERS/JSP1.4.4 シミュレータ

TOPPERS/JSP は、2011 年 5 月 20 日にアップデートが行われ、バージョンが 1.4.3 から 1.4.4 に変更された。これにより、シリアルポート番号の最大値や、変数名などが変更されている。

3.2. 実行環境部のシミュレータ

本論文では、LEGO MINDSTORMS NXT[4]を用いたライントレーサと、ロボットアームの 2 つのシステムをシミュレータの対象とした。

MINDSTORMS NXT とは、プログラムが組み込めるブロックにモータやセンサなどの部品を組み合せることによって、ロボットや他の機械などのシステムを組むことができるレゴ社の商品セットであり、本体となる NXT ブロックには、入力ポートを 4 つ、出力ポートを 3 つ備えており、Bluetooth ワイヤレス接続により、データの通信が可能である [4]。

この節では、試作した実行環境部について説明する。

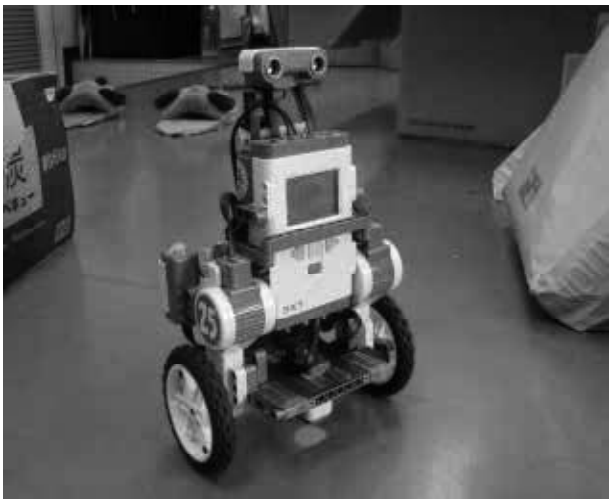

図 6 ライントレーサ

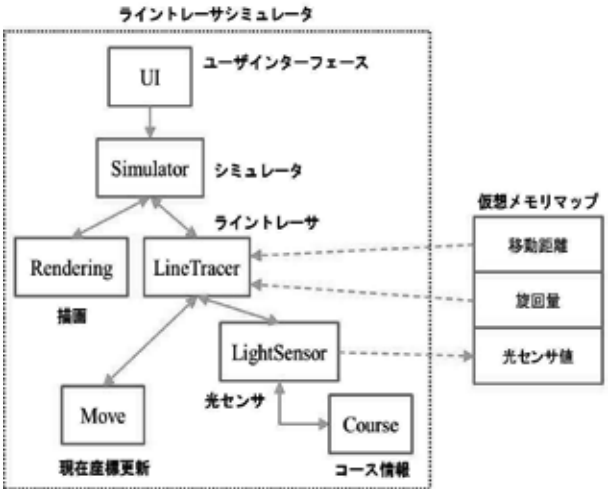

図 7 ライントレーサシミュレータの構成

3.2.1. ライントレーサシミュレータ

ライントレーサとは、床面に描かれた線を検出し、その線に沿って走る走行体のことである。本論文で対象とした

ライントレーサ「NXTway-GS[5)]」を、図 6 に示す。NXTway-GS とは、LEGO MINDSTORMS NXT と HiTechnic 社製のジャイロセンサを用いて作成した二輪倒立振子ロボットのことである[5)]。

このライントレーサは、入力ポートに光センサとジャイロセンサを接続し、出力ポートにモータを 2 つ接続している。赤外線 LED の光を床面に当て、反射した光を光センサで受光し、その結果によって左右のモータを制御して線に沿った走行をする。

この動作を再現するシミュレータを実行環境部として試作した。このシミュレータは、ライントレーサの動く様子を二次元で描画する。また、様々なコースに対応するために、データとしてコースデータを、ビットマップ形式で入力する。

図 7 に試作したライントレーサの構成を示す。以下、それぞれの構成要素について説明する。

- UI

ユーザインターフェースのことである。Simulator に任意のタイミングで開始命令や停止命令を送る。

- Simulator

シミュレータのメインとなる部分である。UI から命令を受け取り、それに合わせた動作を行う。ここから他の構成要素に命令を送信する。

- Rendering

描画する部分である。Simulator から描画命令を受け取り、現在の状態やライントレーサの位置などを描画する。

- LineTracer

ライントレーサに関する情報を保持したり、座標の更新やセンサ値の読み取りの命令を出したりする部分である。また、仮想メモリマップから、旋回量や移動距離を取得する部分でもある。

- Move

ライントレーサが現在いる座標位置を更新する部分である。LineTracer から座標更新命令を受け取り、座標位置を更新して LineTracer に送信する。

- LightSensor

光センサの値を読み取る部分である。LineTracer から一定時間ごとにセンサ値読み取り命令を受信し、Cource に格納している色情報からセンサ値を読み取る。求めたセンサ値は、仮想メモリマップの対応するアドレスに上書きする。

- Course

入力として与えるコースデータを元にして、シミュレートするコースの全ての座標に対する色情報を格納している部分である。

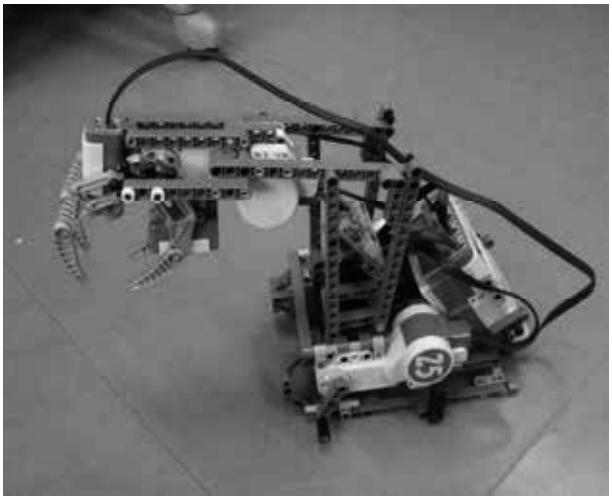

図 8 ロボットアーム

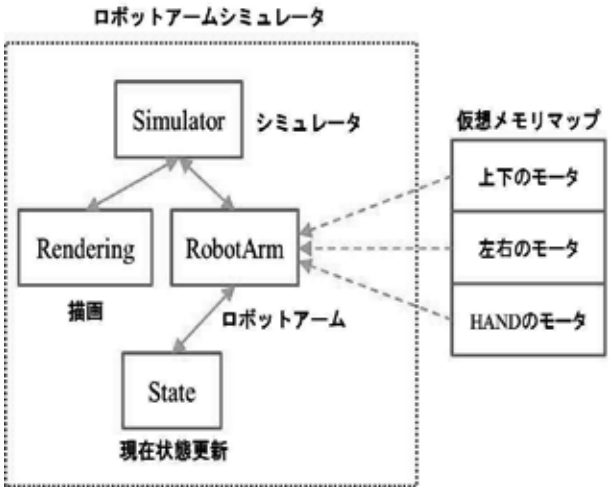

図 9 ロボットアームシミュレータの構成

3.2.2. ロボットアームシミュレータ

ロボットアームとは、人間の手の代わりに、物を運ぶなどの作業をする機械の腕のことである。本論文で対象としたロボットアームを、図 8 に示す。

このロボットアームは、入力ポートには何も接続しておらず、出力ポートに、アームを左右に動かすモータと上下に動かすモータ、ハンドの部分を制御するモータの 3 つのモータを接続している。この動作を再現するシミュレータを実行環境部として試作した。このシミュレータは、ロボットアームの動く様子を 3 次元で描画する。ライントレーサシミュレータとは異なり、入力として与えるデータは無い。

図 9 に、試作したロボットアームの構成を示す。以下、それぞれの構成要素について説明する。

- Simulator

ライントレーサシミュレータと同様、シミュレータのメインとなる部分である。ここから他の構成要素に命令を送信する。

- Rendering

描画する部分である。Simulatorから描画命令を受け取って、ロボットアームの状態を描画する。

- RobotArm

ロボットアームに関する情報を保持し、現在の状態の更新命令を出す部分である。仮想メモリマップから、3 つのモータの回転角度を取得し、State に状態更新命令を送信する。

- State

ロボットアームの状態を更新する部分である。ロボットアームのハンドの状態や、横方向、縦方向への回転角度を更新する。更新は、RobotArm から状態更新命令を受信した時に行い、その結果を RobotArm に送信する。

4. 検証

この章では、今回試作した二部構成シミュレータが実機の動作を正しく再現しているか、また、開発環境部と実行環境部の組合せを変更しても、正しく動作するかについて検証する。

検証方法として、まず、開発環境部にTOPPERS/JSP1.4.3シミュレータ、実行環境部にライントレーサシミュレータを使用し、二部構成シミュレータを作成し、動作確認を行う。その後、この二部構成シミュレータの開発環境部をTOPPERS/JSP1.4.4シミュレータと入れ替えて、動作確認を行う。最後に、実行環境部をロボットアームシミュレータと入れ替えて、動作確認を行う。

4.1. TOPPERS/JSP1.4.3 とライントレーサ

今回試作したライントレーサシミュレータは、一定時間ごとに更新する現在の座標を用いることによって、軌跡を描くことができる。これによって描いた軌跡と、走行体を実際に走行させて描く軌跡を比較する。

使用するプログラムのアルゴリズムは同一のものであり、ライントレーサシミュレータに入力するコースデータは、実際のコースと同比率のものを作成し、使用する。比較には、実機、シミュレータ共にスタートしてから 1 周するまでの軌跡を描いて比較する。図 10 に、実機を用いて走行した場合の軌跡と、二部構成シミュレータ上で実

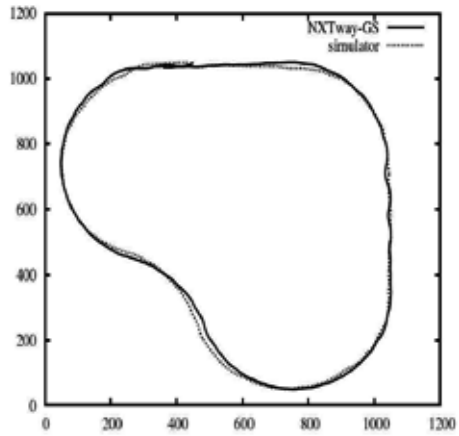

図 10 ライントレーサの軌跡比較
(TOPPERS/JSP バージョン 1.4.3)

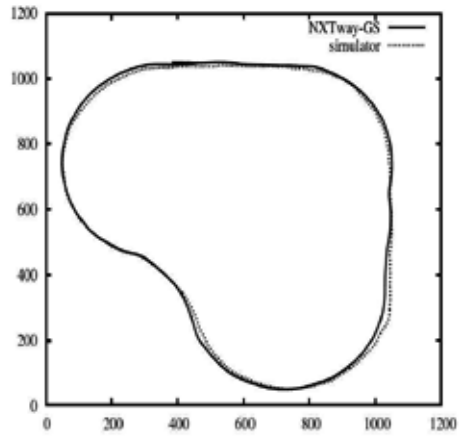

図 11 ライントレーサの軌跡比較
(TOPPERS/JSP バージョン 1.4.4)

行した場合の軌跡を示す。実線が実機の場合で、点線が二部構成シミュレータの場合である。比較した結果、少々誤差はあるものの、大きな誤差はなく、実機に近い軌跡を描くことができていると言える。

この事から、TOPPERS/JSP1.4.3シミュレータとライントレーサシミュレータを組み合わせた二部構成シミュレータは、実機の動作を正しく再現できていると言える。

4.2. TOPPERS/JSP1.4.4 とライントレーサ

次に、4.1節の二部構成シミュレータの開発環境部をTOPPERS/JSP1.4.4シミュレータと入れ替えて、新たに二部構成シミュレータを作成する。

入力するプログラムは、4.1節で初めに使用したプログラムと同じ物を使用し、実行環境部に入力するコースデータも同じ物を使用した。

4.1節と同様、スタートしてから一周するまでの軌跡を描いて比較する。図11に、実機での軌跡と二部構成シミュレータ上で実行した場合の軌跡を示す。4.1節と同様、実機とシミュレータの軌跡には大きな誤差は無く、実機に近い軌跡を描くことができていることが確認できた。これにより、TOPPERS/JSP1.4.4とライントレーサを組み合せた二部構成シミュレータは、実機の動作を正しく再現できていると言える。

この事から、開発環境部のみ変更して実行しても、二部構成シミュレータは正しく動作することが確認できた。

4.3. TOPPERS/JSP1.4.4 とロボットアーム

最後に、4.2節の二部構成シミュレータの実行環境部をロボットアームシミュレータと入れ替えて、新たに二部構成シミュレータを作成する。

このロボットアームは、縦方向、横方向共に、回転できる角度に限界があり、それ以上回転すると、歯車の部分が引っかかってしまい、故障の原因となってしまう。そのため、組込むプログラムはこの角度を超えないように作成しなければいけない。

そこで動作確認として、一定時間右方向へと回転を行った後、停止するプログラムを作成した。この回転する時間を変え、シミュレータ上で限界となる角度を超える時間がどのくらいになるかを検証した後、実際のロボットアームを用いて確認する。図12に、シミュレータ上で右方向へロボットアームが動いた角度と、その時間のグラフを示す。横軸は時間t、縦軸は回転した角度θである。検証はtの値を200ずつ増やし、0~1000までの時間で検証した。実線がその時の回転角度、点線は、ロボットアームが回転できる限界の角度である。これによって得られた角度を線で結ぶと、限界の角度となる時間が、約t=700と推測できる。これと、シミュレータ上で使用したtの値を実際にロボットアームに入れ、確認する。これによって得られた角度のグラフを、図13に示す。ロボットアームが危険と判断したところでプログラムを強制的に終了させているため、これ以上の時間では実行していない。その結果、シミュレータで得られた結果とほぼ同じt=700の所でロボットアームの限界角度付近の角度に達することを確認した。これにより、TOPPERS/JSP1.4.4とロボットアームを組み合わせた二部構成シミュレータは、実機の動作を正しく再現できていると言える。

この事から、実行環境部のみ変更して実行しても、二部構成シミュレータは正しく動作することを確認できた。

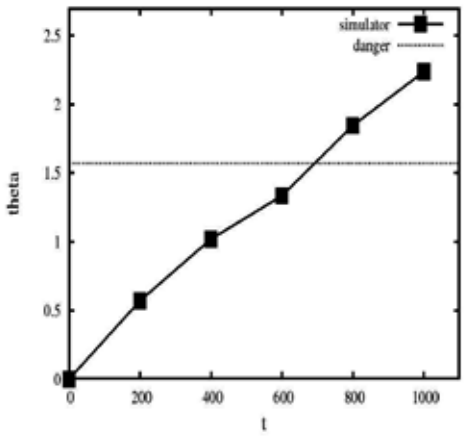

図 12 シミュレータの実行結果

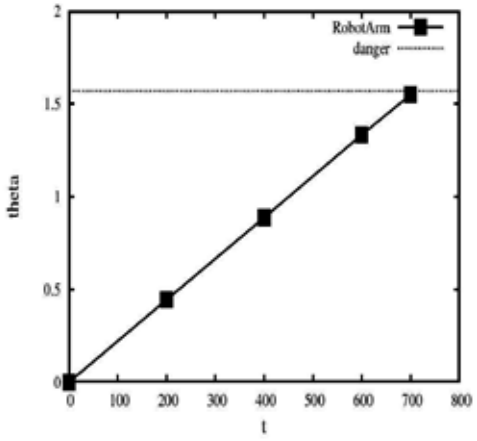

図 13 ロボットアームの実行結果

5. 考察

本研究では、シミュレータ開発の手間の削減を目的として、二部構成シミュレータ設計手法を提案した。この手法で作成する二部構成シミュレータは、開発環境部と実行環境部が並列に動作を行っうことによって、対象となるシステムをシミュレートする。

この章では、この二部構成シミュレータの有効性について、再利用率と関連研究の点から考察する。

5.1. 再利用率

開発環境部および実行環境部として作成したシミュレータのプログラムの行数を調べ、再利用率を求める。

表 1 シミュレータのプログラム行数

開発環境部	プログラム行数
TOPPERS/JSP1.4.3 シミュレータ	10,409
TOPPERS/JSP1.4.4 シミュレータ	10,531
実行環境部	プログラム行数
ライントレーサ シミュレータ	1,764
ロボットアーム シミュレータ	1,285

表 2 プログラムの再利用率

二部構成シミュレータの組合せ	再利用率
TOPPERS/JSP1.4.3とライントレーサ 共通ヘッダ 19行 10,409 + 1,764 +19= 12,192 (行)	
TOPPERS/JSP1.4.4とライントレーサ 共通ヘッダ 19行 10,531 + 1,764+19 = 12,314 (行)	14.3%
TOPPERS/JSP1.4.4とロボットアーム 共通ヘッダ 16行 10,531 + 1,285+16 = 11,832 (行)	89.0%

表1に、それぞれのシミュレータのプログラムの行数を示す。また、表2に、4章で作成した3つの二部構成シミュレータを作成する際の再利用率を示す。

この表より、4.1節の二部構成シミュレータから4.2節の二部構成シミュレータを作成する場合で14.3%、4.2節の二部構成シミュレータから4.3の二部構成シミュレータを作成する場合で89.0%、プログラム行数を再利用できたことが確認できる。

今回の検証の場合、開発環境部と実行環境部のプログラムの行数に差があるため、開発環境部を変更して実行した場合と実行環境部を変更して実行した場合で再利用率が大きく変わっている。しかし、変更しなかった方の部は一切プログラムを変更することなく使用可能であったため、開発環境部と実行環境部のプログラムの行数に差がない場合だと、約50%の再利用率が見込まれる。

5.2. 関連研究

関連研究として、分散シミュレーションがある[6)]。これ

は、ネットワーク接続された複数のシミュレータを連携して動作させる方法である。この方法により、1つのシミュレータを機能分解して複数のコンピュータに分散配置することによって柔軟なシミュレーション構成を可能とする。

分散シミュレーションは、ネットワークで接続されたコンピュータ間でデータの通信を行なっているため、地理的に離れた場所で並列して動作すること可能である。また、分散させたシミュレータは別のコンピュータで動作するため、負荷を分散でき、その結果シミュレーションを高速化することが可能といった利点がある。しかし、ネットワークを介してデータの通信を行うため、長距離通信路を利用する場合は、通信に遅延が発生してしまい、シミュレーションの速度が逆に遅くなってしまう。また、分割したシミュレータそれぞれが別のコンピュータに配置されるため、複数のコンピュータが必要であり、それらのシミュレータは単体で動作できなければならない。

これに対し、本研究で提案した二部構成シミュレータは、ネットワークを介さずに1台のコンピュータ上でデータのやりとりを行うため、分散シミュレーションと比べ、データの遅延が少ない。そのため、データの遅延が致命的となりうるような高いリアルタイム性が求められるシステムをシミュレートする場合には、今回提案した二部構成シミュレータの方が有効であると考えられる。

6. おわりに

本研究では、シミュレータ作成の手間の削減を目的として、二部構成シミュレータ設計手法を提案した。提案手法で作成する二部構成シミュレータは、開発環境部と実行環境部の2つの部で構成しており、それぞれの部が並列に動作を行ってシミュレートする。また、開発環境部と実行環境部の結合度が低いため、新たに二部構成シミュレータを作成する際にそれぞれの部を再利用することが可能である。

これにより、組込みシステムの開発の際に用いるシミュレータを作成する手間を削減することが可能である。これは、組込みシステムの生産性の向上に繋がると考えられる。

以下に、今後の課題を挙げる。

- 他の開発環境部や実行環境部の作成および再利用性の検証

本論文では、開発環境部を実行環境部を、それぞれ2つずつ用意して再利用性の検証を検証した。二部構成シミュレータの有用性を検証するために、今後、新たに開発環境部および実行環境部の作成を行い、それらと今回用いたシミュレータを組合せて再利用性をさらに検証する必要がある。

- シミュレータ上での実行周期変化機能の実装

本論文で試作した二部構成シミュレータは、開発環境部と実行環境部間のデータのやり取りの際の同期を取ることを優先し、シミュレータ上での実行周期を変化させる機能を実装していない。そのため、実行周期を変化させるには、ヘッダに記述している実行周期を変更し、再度実行を行わなければならず、手間がかかってしまう。

そのため、開発環境部と実行環境部間の同期を取りつつ、シミュレータ上での実行周期が変化できるような機能を実装する必要がある。

- 実際の組込みシステム開発への適用および有用性の検証

本論文では、検証のために、シミュレートの対象となるシステムは、単純な物を使用した。

今後、本論文で提案した手法を実際の組込みシステム開発に適用して、二部構成シミュレータの有用性を検証すると共に、問題点を洗い出す必要がある。

参考文献

1) 星 光行: STARC 2010 組込み SW 設計編(C コース)C1 章 組み込みソフトウェアの基礎 1,半導体理工学研究センター STARC,2010.
2) 総務省,統計局: 日本標準産業分類, http://www.stat.go.jp/index/seido/sangyo/19-3.htm.
3) TOPPERS プロジェクト, http://www.toppers.jp/index.html.
4) B. Baqnall: Maximum Lego NXT:Building Robots With Java Brains, Variant Press. 2010.
5) 山本順久:NXTway-GS のモデルベース解暑~LEGO Mindstorms NXT を用いた二輪型倒立振子ロボットの制御~,サイバネットシステム株式会社,2009.
6) 小堀 壮彦,山本 一二三:分散シミュレーション技術の紹介, MSS 技報,Vol.21,pp17-21,2010.