ライントレースカーの動作を反映するシミュレータの試作

池田 健太郎* ・ 片山 徹郎**

Prototype of a Simulator to Reflect Behavior of a Line Trace Car

Kentaro IKEDA, Tetsuro KATAYAMA

Abstract

In embedded systems, it is necessary to confirm behavior of embedded softwares on a real machine. This work takes much time when some bugs exist in the software and it is corrected many times. Then, a simulator to reduce the time is introduced. In this paper, a prototype of a simulator for a line trace car has implemented. By using the simulator, the behavior of the line trace car is reproduced on the simulator and the speed value in the code is read and reflected in the simulator. The improvement of productivity in the embedded system development is expected because the simulator can reduce the time to confirm the behavior of the embedded software on a real machine.

Key Words:
Embedded system, Simulator, Line trace car, Real machine

1. はじめに

組込みシステム技術は日本が世界に誇る技術であり，次世代ユビキタス社会のカギの 1 つである。組込みシステム分野はハードウェアからソフトウェアまで多岐に渡り，最近では国内の大学でも研究が盛んに行われている。

組込みシステム開発を学ぶ際には，ハードウェアやソフトウェアといった枠組みにとらわれずに，常に幅広い視野でシステム全体を見渡し，広い知識と高度な技術で問題を解決する能力が必要となる [1)]。この総合的な問題解決力を身につけるためには，ハードウェアからソフトウェアまでの一連の開発プロセスを実際に体験することが非常に重要である。しかし，学生にとっては，それら一連の開発プロセスを経験する機会が少ないという現状がある。また，組込みシステムという枠組みの中での学生同士の交流も多くはない [2)]。

このような現状を打破するために，2005 年から SSEST[2)] が開催されている。SSEST とは，学生が主体で実施する「組込みシステム技術に関するサマースクール（Summer School on Embedded System Technologies)」のことである。SSEST では，組込みシステムにおける一連の開発プロセスを体験し，システム開発の難しさや面白さを体感することを目的としている。この SSEST には，マイコンの組立て，ロボットの組立て，クロス開発環境による組込みソフトウェア開発といった組込みシステム開発に関する基礎的な要素が随所に含まれている。つまり，SSEST における実習によって，組込みシステム開発の一連の流れについて学べるようになっている。

2006 年に開催された SSEST2 では，床面に描かれた線に沿って走るライントレースカーを製作した。このライントレースカー製作において生じた問題として，実機での動作確認に手間がかかるという点があった。作成したソフトウェアによって実機が期待通りに動くかどうかは，実機上で実際に動作確認を行わなければ分からない。もし期待した通りに動作しなければ，ソフトウェアを修正し，再度動作確認を行う必要がある。実機が期待通りに動くまでには，このような修正作業を何度も繰り返すことになる。

そこで本論文では，実機上での動作確認にかかる手間を削減するために，シミュレータの導入を提案する。シミュレータを導入することで，実機を用いずに動作確認が行える。これにより，実機製作時間を削減する

* 情報システム工学専攻大学院生 ** 情報システム工学科准教授

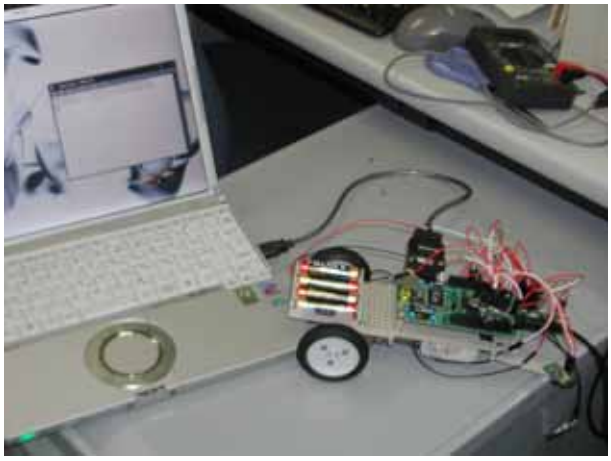

図 1: 製作したライントレースカー

ことができるため，組込みシステム開発における生産性の向上が見込まれる。

なお，本論文では，SSEST2 で実際に製作したライントレースカーを対象とし，

- ライントレースカーの動きをシミュレータ上に再現する。
- コード中の速度値を読み込み，その内容をシミュレータに反映する。

という 2 つの目的を実現するシミュレータを試作する。シミュレータの試作には，特定の環境に依存しない性質を持つプログラミング言語 Java[3] を使用する。また，Java プログラムの統合開発環境として，豊富な機能を持った Eclipse[4] を使用する。

2. ライントレースカー

ライントレースカーとは，床面に描かれた線を検出して走る車のことである。その動作原理は，

i) センサによって，床面に描かれた線を検出する。
ii) 検出した情報を基にモータを制御する。

というものである。図 1 に，SSEST2[2] において実際に製作したライントレースカーを示す。なお，今回製作したライントレースカーには，左右 2 つのセンサがある。

3. シミュレータの構成

図 2 に，今回試作したシミュレータの外観を示す。シミュレータは，「UI(User Interface)」「Simulator」「Mechanism」「LineTracer」「Rendering」「Environment」の 6 つのクラスから構成する。以下，各クラスについて説明する。

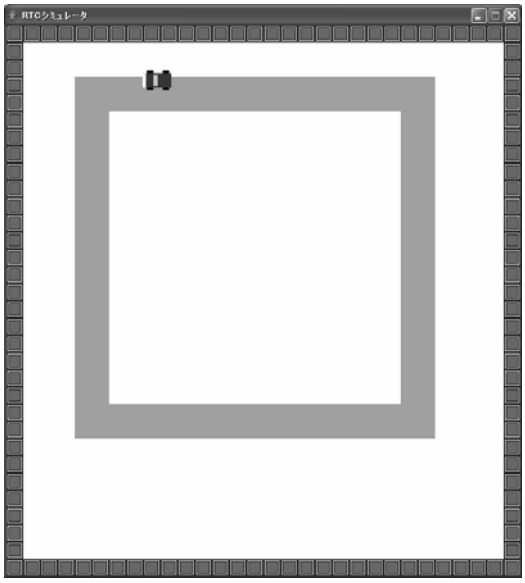

図 2: シミュレータの外観

- UI(User Interface)
  UI クラスは，ユーザからの入力を処理する。ユーザがこの画面上の start ボタンを押すと，シミュレートを開始する。また，stop ボタンを押すとシミュレートを終了する。
- Simulator
  Simulator クラスは，内部の動作を開始または終了する。シミュレート中は動作命令および描画命令を行う。
- Mechanism
  Mechanism クラスは，シミュレータにおけるライントレースカーの現在の座標と角度，および，速度を保持する。また，ライントレースカーの動作が記述してあるコードを読み込む。
- Environment
  Environment クラスは，コース画像の色を保持し，座標の指定に対して黒か白かを表すコース情報を取得する。
- LineTracer
  LineTracer クラスは，コース情報を基にライントレースカーの動作を決定する。
- Rendering
  Rendering クラスは，ライントレースカーの座標と角度に基づいて，シミュレータウィンドウ上にライントレースカーおよびコースを描画する。

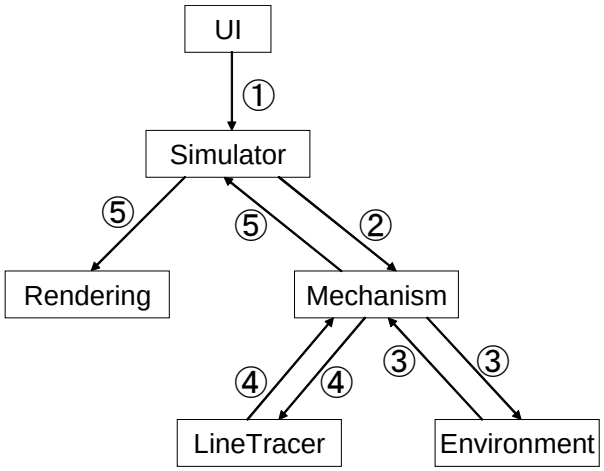

図 3: シミュレータの構成と流れ

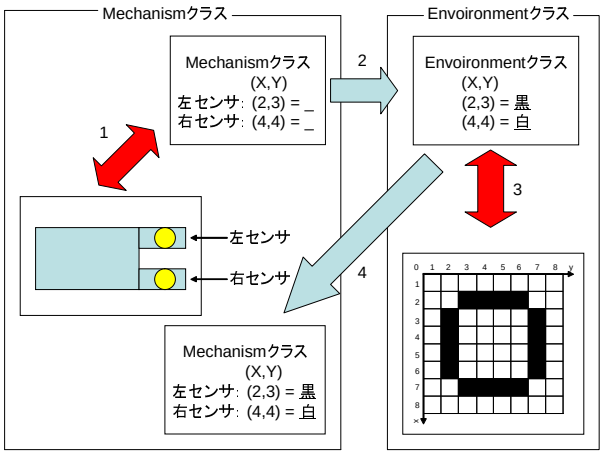

図 4: コース情報取得

3.1 処理の流れとクラス間の関係

図 3 に，今回試作するシミュレータの構成と流れを示す。以下では，シミュレータの流れについて，図 3 を用いて説明する。なお，図 3 における番号 1〜5 は，下記の箇条書き i〜v に対応している。

i) ユーザが GUI(Graphical User Interface) を用いてシミュレータの開始を指示する。このとき UI クラスは，Simulator クラスにシミュレート開始を指示する。

ii) Simulator クラスは，動作の決定を行うために Mechanism クラスを呼び出す。

iii) Mechanism クラスでライントレースカーの座標と角度を設定する。Environment クラスで，センサの座標を基にコース情報を決定する。このコース情報を Mechanism クラスが取得する。

iv) Mechanism クラスが取得したコース情報を基に，LineTracer クラスで，動作を決定する。Mechanism クラスは，決定した動作情報を取得し，ライントレースカーの座標と角度を決定する。

v) Simulator クラスは，Mechanism クラスからライントレースカーの座標と角度を取得する。Rendering クラスは，Simulator クラスからこれらの情報を取得し，描画を行う。

3.2 ライントレースカーの動作確認

ライントレースカーの動きを再現するためには，感知，動作，描画の 3 つの処理を行う必要がある。以下では，この 3 つの処理についてそれぞれ説明する。

3.3 感知

本シミュレータにおける感知とは，Mechanism クラスの指定した座標が黒であるか白であるかというコース情報を，Environment クラスから取得することである。以下では,「コース情報取得までの流れ」と「座標」について説明する。

コース情報取得までの流れ

図 4 は，Mechanism クラスで座標を指定し，その座標を基にコース情報を取得するまでの流れを示している。以下，図 4 に沿ってコース情報取得までの流れを説明する。なお，図 4 における番号 1〜4 は，下記の箇条書き i〜iv に対応している。

i) Mechanism クラスで，シミュレータ上の左センサおよび右センサの (X,Y) 座標を設定する。
例として図 4 では，Mechanism クラスで，左センサの (X,Y) 座標=(2,3)，右センサの (X,Y) 座標=(4,4) と設定する。

ii) Environment クラスが，Mechanism クラスで設定した座標を取得する。
例として図 4 では，Environment クラスが Mechanism クラスから，左センサの (X,Y) 座標=(2,3)，右センサの (X,Y) 座標=(4,4) という情報を取得する。

iii) Environment クラス内で取得した座標を，コース情報に変換する。コース情報は黒か，白の値である。コース情報が黒のとき，センサはコース上であることを示し，白であるとき，センサはコース外であることを示す。
例として図 4 では，Environment クラスで，左セ

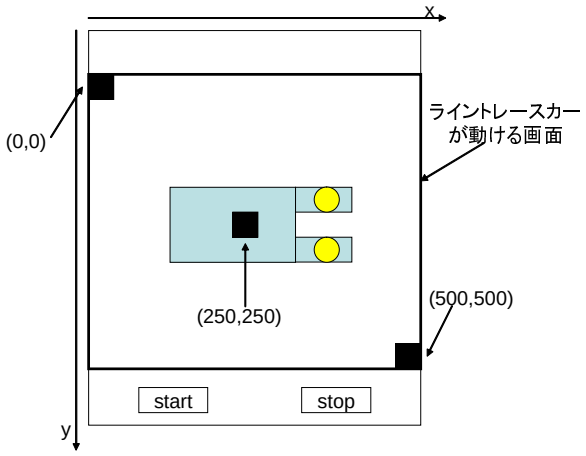

図 5: 座標の概念

ンサの (X,Y) 座標=(2,3)=黒，右センサの (X,Y) 座標=(4,4)=白と変換する。

iv) Mechanism クラスは Environment クラスが，変換したコース情報を取得する。
例として図 4 では，Mechanism クラスが Environment クラスから，左センサの (X,Y) 座標=(2,3)=黒，右センサの (X,Y) 座標=(4,4)=白という情報を取得する。

座標

図 5 に，本シミュレータにおける座標の概念を示す。(X,Y) 座標とは，図の黒い太枠のように，シミュレータウィンドウの中の，ライントレースカーが動ける範囲の座標のことである。今回は，図のように，この中の左上の端点を (0,0)，右下の端点を (500,500) とする。これは，今回使用するシミュレータのウィンドウは，縦 500 ピクセル，横 500 ピクセルの画面であることを意味する。このときのライントレースカーの座標は，図のようにライントレースカーの中心の座標を基にして考えている。

図 5 では，ライントレースカーの座標は (X,Y) = (250,250) となる。これは，左上の端点から，縦に 250 ピクセル，横に 250 ピクセルの場所を意味する。なお，X 座標は左から右へ行くと増え，Y 座標は上から下へ行くと増える。

3.3.1 動作

シミュレータにおける動作とは，Mechanism クラスが持つコース情報を基に LineTracer クラスが動作を決め，ライントレースカーの座標と角度の更新を行うことである。以下では,「角度更新の流れ」「コース情報と動作の対応」「動作の様子」「ライントレースカーの位置および向き」について説明する。

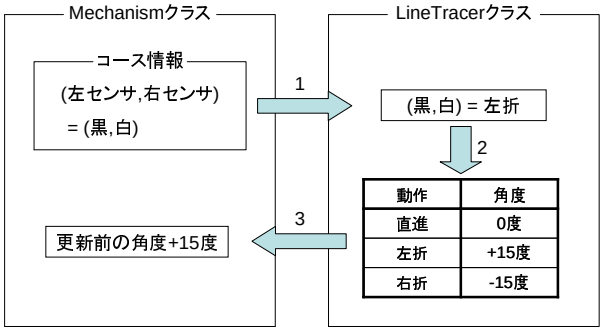

図 6: 角度更新の流れ

角度更新の流れ

図 6 は，コース情報を基に角度の更新を行うまでの流れを示している。以下では，図 6 に沿って角度更新までの流れを説明する。なお，図 6 における番号 1〜3 は，下記の箇条書き i〜iii に対応している。

i) LineTracer クラスが Mechanism クラスからコース情報を取得する。
例として図 6 では，LineTracer クラスが Mechanism クラスから，(左センサ, 右センサ)=(黒, 白) という情報を取得する。

ii) LineTracer クラス内でコース情報を基に動作を決定する。
例として図 6 では，まず LineTracer クラスで，(左センサ, 右センサ)=(黒, 白)=左折と設定する。次に，左折の場合は，角度変化= +15 度と設定する。

iii) 決定した動作を基にライントレースカーの座標および角度を決める。
例として図 6 では，Mechanism クラスが LineTracer クラスから，角度変化= +15 度という情報を取得する。この情報を基に，Mechanism クラスで，更新前の角度+15 度とする。ここで新たに設定した角度を基に，ライントレースカーの座標を更新する。設定した角度を基に，ライントレースカーの座標を決める方法は，後述する。

コース情報と動作の対応

表 1 に，今回試作したシミュレータにおける，センサの感知した値，それを基に決定した動作，動作を行

表 1: コース情報による動作

左センサ	右センサ	動作	角度
黒	黒	直進	0
黒	白	左折	+15
白	黒	右折	−15
白	白	回転	+15

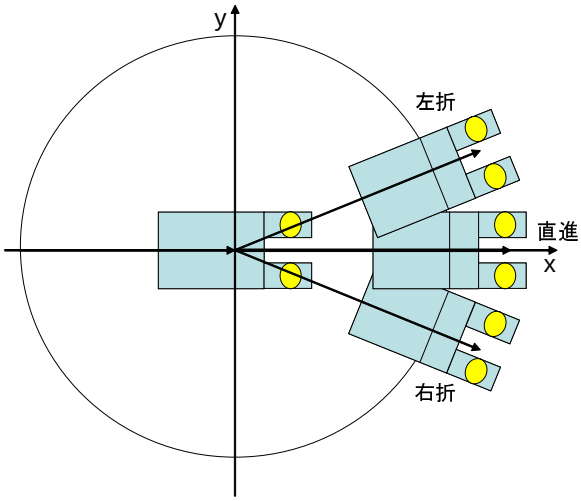

図 7: シミュレータの動作の様子

う際の角度変化を示す。LineTracer クラスは，左センサおよび右センサのコース情報を基に動作を決定する。以下，表 1 について説明する。

- 左センサが黒，右センサが黒の場合
  シミュレータ画面におけるライントレースカーは，直進する。直進の場合，角度変化はないとする。
- 左センサが黒，右センサが白の場合
  シミュレータ画面におけるライントレースカーは，左折する。左折の場合，角度変化は+15 度とする。
- 左センサが白，右センサが黒の場合
  シミュレータ画面におけるライントレースカーは，右折する。右折の場合，角度変化は −15 度とする。
- 左センサが白，右センサが白の場合
  シミュレータ画面におけるライントレースカーは，回転する。回転の場合，角度変化は+15 度とする。
  なお，シミュレータにおけるライントレースカーの回転とは，左折を繰り返すこととする。

動作の様子

今回試作したシミュレータには，ライントレースカーのように，モータの制御による車輪の回転という概念はない。代わりに，決定した動作に基づき，本体が位置を変化させ，向きを変えるという概念を用いている。なお，コースからはずれた場合に，コース上に戻るように向きを変えるという考え方 ([3.2 節センサの検出値による動作] 参照) は，ライントレースカーと同じである。しかし，シミュレータではモータを制御するという考え方ではなく，本体が回転するという考え方をしている。図 7 に，シミュレータの動作の様子について示す。以下，図 7 を基に，動作の様子について，表 1 を参考にしながら示す。

- 直進の場合は，図 7 の直進のように角度を変えずに走行する。
- 左折の場合は，図 7 の左折のように角度を+15 度変えて走行する。
- 右折の場合は，図 7 の右折のように角度を −15 度変えて走行する。
- 回転の場合は，図 7 の左折と同じように角度を+15 度変えて走行する。
  なお，シミュレータにおけるライントレースカーの回転とは，左折を繰り返すことである。

ライントレースカーの位置および向き

Mechanism クラスが LineTracer クラスから得る情報は，角度変化だけである。しかし，ライントレースカーは，初期座標，初期角度，1 回の移動距離を設定しておけば，角度変化だけで，動作後の位置座標と向きが一意に定まる。これは，以下の [定義] のように座標，角度，移動距離を設定したとすると [式] (1)〜(3) が成り立つからである。実際に数値を定義した場合の例を，[例] に示す。また，図 8 は，[例] で設定した値を基に，ライントレースカーの位置および向きが変化する様子を示す。

[定義]

- R：1 回の動作における移動距離
- X1：更新前の x 座標
- Y1：更新前の y 座標
- $\theta 1$：更新前の角度
- α：動作の決定により変化する角度

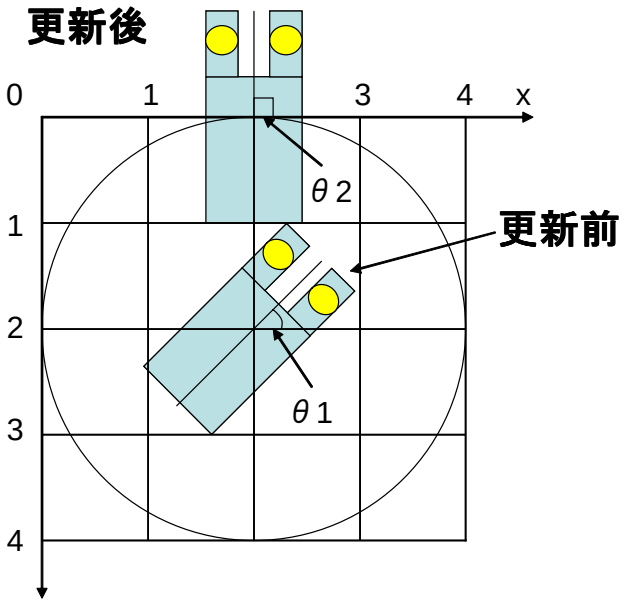

図 8: 位置更新の例

- X2：更新後の x 座標
- Y2：更新後の y 座標
- $\theta 2$：更新後の角度

[式]

(1)$\theta 2=\theta 1+\alpha$

(2)X2=X1+Rcos$\theta 2$

(3)Y2=Y1−Rsin$\theta 2$

[例]

初期条件として以下の (1)〜(3) を与えるとする。

(1) 初期座標 (X1,Y1)=(2,2)

(2) 初期角度 $\theta 1$=45 度

(3) 1 回の移動距離 R=2

このとき，図 8 では，更新前のライントレースカー (X,Y) 座標=(2,2)，角度=45 度を示す。

変化する角度 α=45 度とすると，$\theta 1$ と α が分かるので，[式](1) より $\theta 2$=90 度と分かる。$\theta 2$ が分かったので，X1=2，Y1=2，R=2，$\theta 2$=90 度を，[式](2)，[式](3) に代入すると，

X2=2+2cos90 °

X2=2

Y2=2−2sin90 °

Y2=0

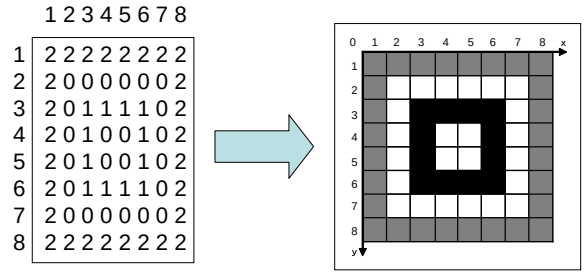

図 9: コース画像の作成

となり，更新後の座標と角度が分かる。図 8 では，更新後のライントレースカー (X,Y) 座標=(2,0)，角度=90 度を示す。

3.4 コースの表示

今回のコースは，画像を読み込んで表示を行うのではなく，数値による 2 次元配列を変換することで画像を作成し，表示を行う。これは，画像の色を読み込み，その値を基に動作を行うよりも，処理が簡単で，表示も早いと考えたからである。また，あらかじめ画像と色を対応させておくことも可能である。

図 9 に，データ画像をコース画像に変換する様子を示す。図 9 のように，今回作成する数値による 2 次元配列は，0，1，2 から構成する。このデータは，0=白い画像，1=黒い画像，2=壁画像がそれぞれ対応している。白い画像は，コース外であることを意味する。左右のセンサは白い画像上にあった場合，向きを修正する。黒い画像は，コース上であることを意味する。左右のセンサが黒い画像上に位置するように走行する。壁画像は，その画像にぶつかるとライントレースカーが停止することを意味する。この 3 種類の画像によりシミュレータ画面を構成する。

3.5 コードの動作確認

今回試作するシミュレータでは，シミュレータ上のライントレースカーに，コード内容における速度値の反映を行う。なお，コードの読み込みは，Mechanism クラスが行う。まず，ライントレースカーの動作が記述してあるファイルを読み込み，その内容から，速度値を取得する。今回のライントレースカーの動作が記述してあるコードにおいて，速度値は，変数「MOTOR_STOP_RATE」に格納される。そこで，String 型の関数を用いて，この変数名と一致する文字列を調べることによって，この変数の値を取得し，速度値を得る。次に，取得した速度値によりシミュレータ上の速度値を更新する。これにより，シミュレータ上のライントレースカーの速度が変化する。なお，これらの

処理はあらかじめ読み込むファイルを決めておき，シミュレータが起動したときに自動的に実行する。つまり，シミュレータ起動後に読み込むファイルを指定することはできない。

4. 実行例

本節では，試作したシミュレータがライントレースカーの動作を正しく再現しているか，また，ライントレースカーの動作が記述してあるコードを読み込ませると，シミュレータ上のライントレースカーがコード内容を反映して正しく動作するかを確認する。

4.1 動作確認

今回のシミュレータにおけるユーザからの入力は，start ボタンと stop ボタンを押すだけであり，出力として，画面上にコースとライントレースカーが表示される。これは，速度値を読み込んで動作させた場合も同じである。また，今回使用したコースは，直径約 50cm の円形のコース 1 周である。

表 2 に，シミュレータの周回速度を示す。なお，速度値の大きさは，コードにおける値であり，値が大きくなると動作が速くなる。

表 2 の値は，実機の周回速度の値とは異なっており，実機の動作を十分に表現することはできていない。これは，今回試作したシミュレータでは，ライントレースカーがどのような動きをするかは確認できるが，その動きは実機の動作に正確には対応できていないためである。今回の動作確認において確認できたことは，シミュレータ上のライントレースカーがコース情報を感知し，その情報を基に動作を行う様子である。

4.2 コード確認

今回は，ライントレースカーの動作が記述してあるコードから速度値を読み込み，その値をシミュレータの動作に反映させる。表 2 に，コードの速度値を変更した際のシミュレータの周回速度を示す。

表 2 の値は，実機の周回速度の値とは異なっており，実機の動作を十分に表現することはできていない。しかし，速度値を変更した場合は，シミュレータ上のライントレースカーの速度も変化した。このことから，コードから読み込んだ速度値を，シミュレータの動作に反映していることが分かる。

5. 考察

本研究では，組込みシステム開発における動作確認およびコード確認を目的とし，シミュレーターを試作した。なお，今回試作したシミュレータは，SSEST2[2] にて製作したライントレースカーを対象としている。

表 2: 動作確認およびコード確認においての周回速度 (sec)

	シミュレータ
周回速度 (速度値 2)	36.172
周回速度 (速度値 3)	24.344
周回速度 (速度値 6)	12.985

本シミュレータ上で，ライントレースカーが，コース情報の取得を行い，コース状況に応じて走行する様子を確認できた。また，実機のコードにおける速度値を反映できた。

今回試作したシミュレータが動作確認の観点で実現できたことは，実機の動作の目安として，ライントレースカーがどのような動作をするか確認できるということである。また，コード確認の観点で実現できたことは，コードにおける速度設定の目安として使用できるということである。

以上から，今回試作したシミュレータは，ライントレースカーの動作の把握，速度設定時の目安として実機製作を支援できると言える。このことから，実機製作時間の削減が可能であり，組込みシステム開発における生産性の向上につながると考えられる。

本研究で試作したシミュレータの問題点について，以下に挙げる。

- 実機に近い動作の実現
  今回試作したシミュレータ上のライントレースカーは，実機と周回速度が異なる。このことを踏まえて，再度ライントレースカーを分析し，シミュレータの改良を行う必要がある。

  具体的には，サンプリング周期及び角度変化の測定，速度及び加速度の測定，コース及び車体の大きさの測定を行う必要がある。サンプリング周期及び角度変化の測定により，サンプリング周期を変化させた場合の速度及び回転の際の角度変化をシミュレータ上に反映する。速度及び加速度の測定により，シミュレータ上で実機と常に同じ動作を行うようにする。コース及び車体の大きさの測定により，シミュレータ上に実際の実機の動きと同じ状況を再現する。これらの測定により，実機に近い動作の実現ができると考えている。

- 速度値以外のコード内容の反映
  今回試作したシミュレータでは，コード内容における速度値のみ反映している。しかし，速度値の

反映だけではシミュレータ上で全コード内容を反映させたライントレースカーの動きを確認することはできない。

そこで，この問題を解決するために実機のソースコード内容を構文解析し，その内容をシミュレータ上のライントレースカーに反映する方法が考えられる。これは，実機のソースコードを構文解析し，その内容をシミュレータ上のライントレースカーの動作に反映する方法である。

また，実機におけるマイコンの実装をする方法も考えられる。これは，実機におけるマイコンの役割となる部分の実装により，解析したソースコードの内容をシミュレータ上のライントレースカーに反映することである。これらの方法により速度値以外のコード内容の反映ができると考えている。

- シミュレータの時刻変化機能
  今回試作したシミュレータは，シミュレート上の実行時間と現実時間を対応させて作成した。このため，現実にかかる時間と同じ程度の時間でシミュレートした結果を得ることができる。シミュレータの結果を早く得たい場合には，シミュレータの時刻変化を現実時間より速く動作させることで対応できる。また，動作をゆっくり見たい場合には，遅く動作させればよい。

  この時刻変化機能を実装するためには，シミュレート結果の再生速度を変化できるようにする必要があると考えている。

- 実際の組込み開発への適用
  今回作成したシミュレータの機能は，それほど豊富ではない。しかし，シミュレータを組込みシステム開発に導入することにより，実機上での動作確認にかかる手間を削減することができる。また，実機が完成していない状況でも，ソフトウェアの検証が出来る可能性がある。さらに，コードを1ステップずつ実行させるなどの時間制御が容易になり，過負荷などの異常状態を仮想的に発生させることも可能であると考えている。

  これらのことを実現するためには，全コード内容をシミュレータ上に反映させる必要があると考えている。また，実際の組込み開発への適用を考えた場合，シミュレータを導入した場合に開発時間をどの程度削減できるかを検証する必要もあると考えている。

6. おわりに

本研究では，組込みシステム開発における動作確認およびコード確認を目的とし，シミュレーターを試作した。本シミュレータでは，SSEST2[2)] にて製作したライントレースカーを対象とし，その動作をシミュレータ上で表現した。また，コードから速度値を読み込み，シミュレータに反映した。

今回試作したシミュレータを，実機の動作確認の目安として使用できる。これは，実機上での動作確認時間の削減につながる。また，試作したシミュレータを，コードにおける速度設定の目安として使用できる。これは，ソフトウェアの修正作業時間の削減につながる。よって，本シミュレータを使用することで実機製作時間の削減が可能である。これは，組込みシステム開発における生産性の向上につながると考えられる。

以下に今後の課題を挙げる。

- 実機に近い動作の実現
- 速度値以外のコード内容の反映
- シミュレータの時刻変化機能
- 実際の組込み開発への適用

参考文献

1) 「組み込みソフトウェア　開発スタートアップ」, Design Wave Magazine 2005年7月増刊号, CQ出版社 (2005).

2) SSEST(Summer School on Embedded System Technologies) , http://www.ertl.jp/SSEST/top/index.php

3) David Flanagan(イデア コラボレーションズ株式会社 訳),「JAVA クイックリファレンス 第4版」, 株式会社オライリー・ジャパン (2003).

4) Eclipse project , http://www.eclipse.org/