ヒューマノイドのための新しいロボットプログラミング環境: EusDyna

マイクロカーネル方式による動力学シミューレーション・プログラミングの統合の実現

○小倉崇 岡田慧 稲葉雅幸 (東京大学)

EusDyna: Interactive Dynamics Simulation Robot Programing Environment by Micro-Kernel Simulation Method

*Takashi OGURA, Kei OKADA, Masayuki INABA (The University of Tokyo)

Abstract—
This paper proposes the new robot programming environment in which robot motion programming environment and simulator are integrated. This allows robot motion programs to include simulation descriptions. Additionally, a new implementation of simulation that is composed by simulation modules is presented, on the other hand, conventional simulators are monolithic and implanted every function. This makes it difficult to add new sensors on a simulator by its users. In the new method, the users of the environment can add new modules easily. The system is evaluated by showing new robot motions like brooming, seesaw and so on.

Key Words: Robot Programing, Dynamics Simulator, Humanoid, EusLisp

1. はじめに

Fig.1 に示すのは本稿で紹介する新しいロボットプログラミング環境 EusDyna により実現されたヒューマノイドロボットの行動シミュレーションの例である．このように様々な環境と接触しながらの行動はこれまで実現されてこなかった．これまでも実ロボットでの実験コスト削減のために動力学シミュレータを用いた開発が行われてきている [1] が，床面や把持対象物など，静的で少数の環境下におけるものが多数であった．しかし，ロボットの行動範囲の拡大に従い，環境が多様化しており，モデリング，パラメータ，機能の埋め込みが繁雑になってきている．そのため，これまで **Fig.**1 に示したような床面以外と接触するような環境におけるシミュレーションはほとんど行われてこなかった．

これまでのシミュレーションを利用した行動プログラミング開発は，**Fig.**2 左に示すように，行動プログラミング環境とシミュレータの 2 つの環境を繋ぐことで実現してきていた．そのためシミュレーション世界の行動プログラムからの制御が困難でバッチ的シミュレーションしかできない，動的なシミュレーションパラメータの設定が困難であり，これからのヒューマノイドのシミュレーションには不向きなシステムになっている．

本稿ではまず動力学シミュレーションの行動プログラミング環境への統合について述べ，次にその環境で拡張性，汎用性を持たせるためめの，マイクロカーネル方式のシミュレーションについて述べる．そして，その実装としての EusDyna について説明し，その環境を用いて実現した，ヒューマノイドロボットの環境との相互作用を含む行動シミュレーション例を示すことでこの環境の評価とする．

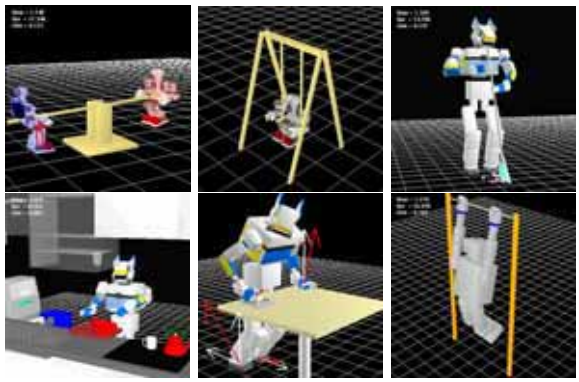

Fig.1 Realized Motions of Humanoid Robots by EusDyna(with See-Saw, Swing, Broom, Kitchen, Sweep and Horizontalbar)

2. 動力学シミュレータ統合型プログラミング環境

動力学シミュレータを行動プログラミング環境へ統合することのメリットは大きくは以下の 3 つであると考える

- 単体でセンサベースプログラムの記述が可能
- シミュレーション世界の調整を行動プログラムから可能
- バッチ処理的ではなく，インタラクティブにシミュレーションを行なうことが可能

まず，単体でセンサベースプログラムが書けるということは，これまでシミュレータで行なわれることが多い行動学習の研究 [2] のために，煩雑なシミュレータとの接続，同期といった必要がなくなり，実験システム

として簡便なものとなる．また，シミュレーション世界は摩擦，衝突のバネ係数などのパラメータは必ずしも汎用的に設定ができるわけではなく，これまであまり行われてこなかった床以外の環境を含んだシミュレーションを行う場合，特に行いたいシミュレーションによって調整が必要になる場合が多くなる．そのため行動プログラムとシミュレータが一体になっていれば，その整合性を取ることが容易になる．また，通常シミュレーションは初期状態からはじまり，ある行動を行なわせると終了するといったバッチ処理的なシミュレーション，もしくは，高速なシミュレータを実機の代わりとして利用するといった利用法がある．行動プログラムと一体とすることで，本当にシミュレーションを必要とする場面でのみ利用するといった使い方が可能になり，効率のよい開発が可能になる．つまり，安全性がすでに確認されている部分に関してはシミュレーションを省略し，現在開発中のコード部分のみをシミュレータにかける，といった使い方が容易に実現できる．

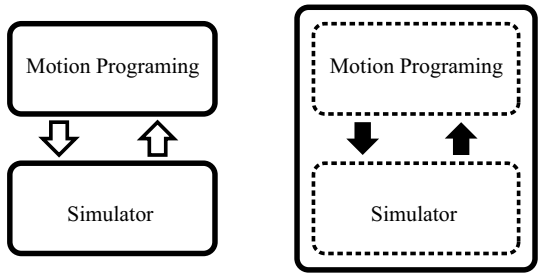

Fig.2 Conventional Development Environments and Proposed Environments that is Integrated with Dynamics Simulator

3. マイクロカーネル方式シミュレーション

本節では，シミュレータの汎用性を高めるための仕組である，マイクロカーネル方式シミュレーションについて述べる．シミュレータは，積分計算，衝突計算，サーボ，各種センサなどのシミュレーションを行う．通常，これらは 1 つのループで表現され，分割，追加には再コンパイルが必要であり，ユーザが容易にできるものではない．この従来方式をコンピュータの OS の実装方式になぞらえ，モノリシックカーネル方式シミュレーションと呼ぶことにする．シミュレーションのコアが 1 枚岩になっているシミュレータの実装方式のことである．これに対し，本稿で提案するのは，システムでは枠組と最低限の機能だけを用意し，ユーザ側で大部分の機能をモジュールとして追加・削除が可能なシミュレーションの実装方式，マイクロカーネル方式シミュレーションである．この 2 つの方式の違いを模式的に表わしたものが **Fig.**3 である．シミュレーションは前述のように，積分計算，衝突計算，サーボ，各種センサなど，機能ごとに計算の種類が分かれている．そこで，この機能ごとにモジュール化し，毎ループごとに各モジュールを呼びだす形にする．ヒューマノイドや知能ロボットといった，頻繁な機能の追加・修正がおこりえるロボットの場合，とくにセンサの追加が容易であるといった利点がある．

一方で，モジュールが増えてゆくとその管理と行動プログラムとの組合せが複雑になることが考えられる．しかし，シミュレータ統合型プログラミング環境においては，行動プログラムにすべて一括して記述することが可能なため，この心配がなくなる．

また，単なる物理現象でない，スイッチの On/Off による電機製品の動作といった，日常環境の動作をシミュレータに取り入れるといったことが今後のシミュレータに求められるようになると考えられるが，このような場合でも，このようなシミュレーション機能をモノリシックカーネル方式のシミュレータであらかじめすべて用意することは困難であるが，マイクロカーネル方式ならば，適時モジュールとして埋め込むことが可能である．

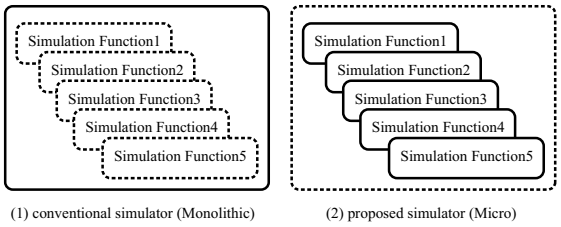

Fig.3 Implements of Conventional Simulator (Monolithic) and Proposed Method(Micro-Kernel)

4. マイクロカーネル方式シミュレーションを利用した動力学シミュレータ統合型プログラミング環境の実装法

4·1 EusDyna の概要

たとえ統合型環境を構築しても幾何・物理モデルをファイルとして別に管理した場合，整合性を取ることが困難になる．また，対話的に操作できることが効率的なデバッグのためには必要である．以上を踏まえ，これまでに述べた，シミュレーション一体型環境，マイクロカーネル方式シミュレーションという特徴に加え，EusDyna は以下の特徴を持つ環境をめざした．

- インタプリタを持つスクリプト言語
- 3D 幾何モデラを持つ
- 物理演算ライブラリ切替可能

他に，バックグラウンドでシミュレーションを行うためのマルチスレッドな環境，実機とのインタフェースをそろえることなどが必要である．これらを実現するために EusLisp[3] をもとにした環境を構築した．EusLisp は松井 (当時電総研) らが開発した幾何モデラ機能を持ちマルチスレッドに対応した Lisp 処理系である．EusLisp は上記の特徴のうち，対話型言語，幾何モデラを持つという特徴を持った言語である．EusLisp の幾何モデラ機能で作成されたモデルに物理特性を与えることで，そのままシミュレーションモデルとした．これまでに EusLisp により多くの行動プログラムが記述されてきており，その資源をそのまま利用することができている．

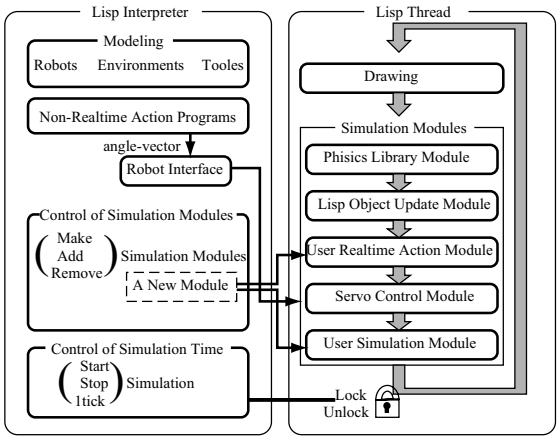

Fig.4 Configuration of EusDyna: Users make models of robots or environments, motion programming and control simulation from interpreter. The thread draws the results of simulation and executes each modules. When users control simulation, the system uses mutex lock.

EusDyna のシミュレータ部分はマイクロカーネル方式により実装されている．**Fig.**4 に EusDyna のインタプリタとスレッドでそれぞれどのようなプログラムが動くかを示す．スレッドでは **Table** 1 に示すようなモジュールが 1 ステップごとに順番に 1 ループ分実行され，シミュレーションが行われる．これはインタプリタからコントロール可能で，1 ステップごとに実行することもできる．シミュレーションのコア計算は外部ライブラリを持ちいてるが，それもひとつのモジュール (Phisics Library Module) として実装されているため，交換が可能である．現在利用可能な動力学計算ライブラリはオープンソースな ODE[4], そのもととなっている Vortex 社の Math Engine, 物理演算ハードウェア (PPU) との連携が可能な PhysX[5], 外部ライブラリに頼らない EusLisp による実装の 4 つがあり，これらの切替が自由に可能になっている．これらのライブラリは完全にカプセル化されており，ユーザはその違いを意識することなくライブラリを使い分けることができる．シミュレーションカーネルとして用意したものは，モジュールの管理機構，そのスレッドでの実行，Viewer へのインタフェースである．

Table 1 Examples of simulation modules

モジュール名	機能
Dynamics	衝突・積分計算．Simulation のコア.
Servo	サーボモータの関節角度を制御
Force Sensor	力センサに発生する力を計測
Accel Sensor	加速度センサ
Gyro Sensor	角加速度センサ
ZMP Sensor	ZMP センサ

4·2 EusDyna の API とコード例

EusDyna の主な API を **Table** 2 に示す．特に特徴的なものはシミュレーションを 1 ステップごと進ませるもの (d-tick)，シミュレーション対象とする幾何形状モデルを指定するもの (d-make)，衝突をさせないペアを指定する (d-no-collision)，空間に固定する (d-fix) などの利用が行動プログラムを記述する上で必要になってくる．

Table 2 Examples of EusDyna API

関数名	役割
d-init	simulation の初期化
d-end	simulation の終了
d-tick	simulation を 1 ステップだけ進ませる
d-start	simulation スレッドを動かす
d-backtick	simulation を 1 ステップもどす
d-make objs	objs にダイナミクスを与える
d-make-joint b1 b2	b1 と b2 の間にジョイントを作成する
d-no-collision objs	objs 間での衝突を無視する
objects objs	objs を表示する

サンプルコードを **Fig.**4·2 に示す．まず，(make-cube) で直方体を 2 つ作成し，色と重量を設定している．その後:locate メソッドにより位置を決めている．この例では (d-init)，(make-dyna), (d-start), (objects) 以外は EusLisp と完全に互換である．それぞれ，初期化，ダイナミクスを与える物体の選択，シミュレーション開始，描画対象の選択を行なう関数である．このコードにより，モデルファイルなどを用意することなく，**Fig.**6 に示すようなシミュレーションを行うことができる．

```
1  (setq aa (make-cube 100 100 100))
2  (setf (get aa :face-color) :blue)
3  (setf (get aa :weight) 50)
4  (setq bb (make-cube 200 200 150))
5  (setf (get bb :face-color) :yellow)
6  (setf (get bb :weight) 60)
7  (send aa :locate #f(50 50 600))
8  (send bb :locate #f(0 0 800))
9  (objects (list aa bb)) ;; 表示
10 (d-init) ;; シミュレーション初期化
11 (make-dyna (list aa bb)) ;; ダイナミクス化
12 (d-start) ;; シミュレーション開始
```

Fig.5 An Example of EusDyna's code

5. 実現した行動プログラミング

これまでに人間とのインタラクションをこのシステムを用いて表現する [6] などの行動シミュレーションを実機と互換性を保ちつつ実現してきているが，EusDyna により新たに実現したシミュレーションでの行動例を **Fig.**1 に先に示した．これを左上から順に解説する．2 台のヒューマノイドがシーソーをしている例では，そ

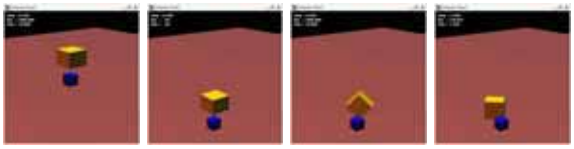

Fig.6 Simulation Results by the Example code

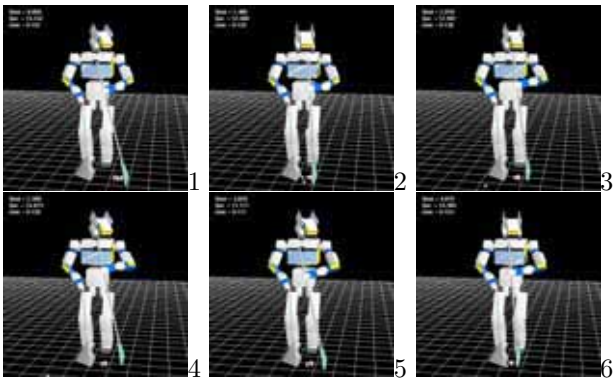

Fig.7 Garbage Collection Motion using Dustpan and Broom

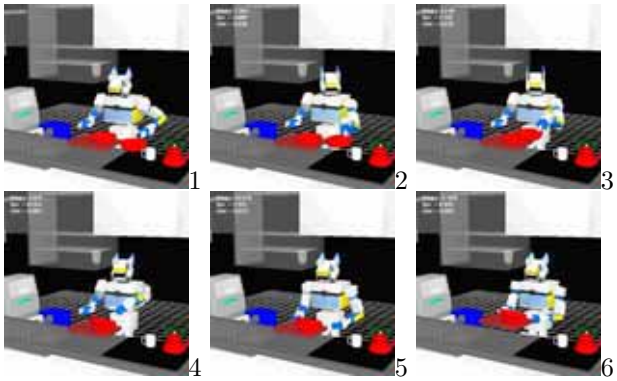

Fig.8 Tidying up Motion (stack the dishes and carry them using a tray) in a Kitchen

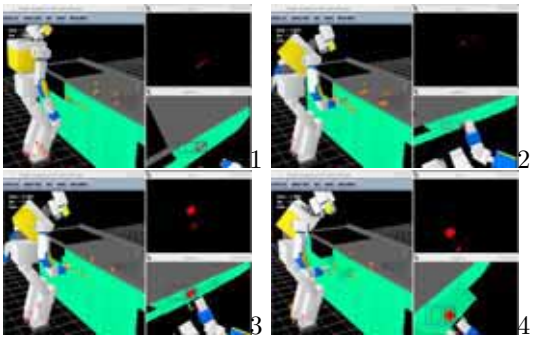

Fig.9 Integration with Vision Processing: Pull the knob recognizing the position by Stereo Vision (The right upper picture is result of color extraction, Left lower one is result of stereo vision processing in each pictures.)

れぞれのロボットが足裏の力のセンサにより蹴り動作を行なうことでうまくシーソー動作ができている．足裏に示した赤い矢印は力センサの値を示しており，この力センサはモジュールとして実装されている．次のブランコの例では強化学習により，体の加速度によりどのように行動を行なえばうまく加速が可能かを学習している．ほうきを掃く動作ではゴミをほうきで掃き，ゴミが実際に集められる様子がシミュレーションできている．ほうきを持たせた状態からシミュレーションを開始することで容易に記述が可能である．ちりとりも持たせたシミュレーションの様子を **Fig.7** に示す．**Fig.**8 に台所におけるシミュレーションの様子を示す．このような複雑な環境でも必要な部分のみダイナミクスを与えることで比較的高速なシミュレーションができる．具体的には，皿 2 枚，作業台，トレイをシミュレーション対象とし，それらを重ね，トレイを持ち，運ぶことができている．また，台を左手で押さえることで体を支えながら，布巾で拭く動作で，手先にかかる力を監視しながらの行動の記述ができている．他に，鉄棒をヒューマノイドが行なうシミュレーションなど，これまでに行なわれたことのない動作例を示すことができている．また，視覚処理との統合もなされており，**Fig.**9 に引き出しを引く例を示す．図中のオレンジ色の矢印は衝突により発生している力を表している．視覚処理を行う部分は 1 つのモジュールとして実装してある．

6. おわりに

これまでにない新しいプログラミング環境として，対話型で幾何モデラを持ち，モジュール型シミュレーションを可能にした EusDyna を構築した．この環境により，これまで行なわれたことのない，数々の行動例を示すことでこの有効性を示した．問題点としては，モジュール化により，計算効率が下がることが挙げられる．しかし，CPU の高速化，PhysX に見られるようにハードウェアによる演算などが進めば，実時間程度のシミュレーションならば問題なく行うことができるようになると考えられる．計算能力の効率化よりむしろ，開発効率を上げるためには本稿で述べたような仕組を導入すべきである．

参考文献

[1] Fumio Kanehiro at el. Open architecture humanoid robotics platform. In *ICRA'02*, pp. 24–30, 2002.

[2] Fumio Kanehiro, Masayuki Inaba, and Hirochika Inoue. Action acquisition framework for humanoid robots based on kinematics and dynamics adaptation. In *ICRA'99*, pp. 1038–1043, 1999.

[3] 松井俊浩．幾何モデリング機能を備えたマルチスレッド並列オブジェクト指向言語 EusLisp．日本ロボット学会誌, Vol. 14, No. 5, pp. 650–654, 1996.

[4] Open Dynamics Engine ODE. `http://ode.org`.

[5] PhysX. `http://www.ageia.com`.

[6] 小倉崇, 岡田慧, 稲葉雅幸．ヒューマノイドの身体誘導行動のためのインタラクティブ動力学シミュレーションシステム．第 5 回 SICE システムインテグレーション部門講演会講演概要集, pp. 1E3–5, 12 2004.