special

Linuxでロボットを作る

最終回 歩行ロボット制御システム

熊谷正朗
kumagai@emura.mech.tohoku.ac.jp

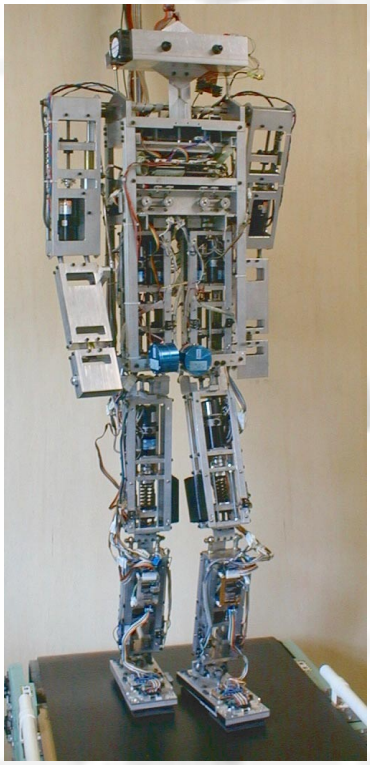
はじめに

第1回はLinuxの普通のプロセスからハードウェアを制御するための方法を、そして第2回はハードウェアよりのデバイスドライバを作成する方法を解説しました（記事末のRESOURCE［1］、［2］を参照）。今回はそれらの内容もふまえて、実際のロボット制御システムとロボットが歩くために必要な処理内容を解説していきます。

ロボットを動かすためには、かなり大まかに分けて2つの要素があります。1つは動かし方（制御理論）*1で、もう1つは動くものを作ることです。シミュレーション研究では、動かし方のみを考えますが、実在のロボットを動かそうとすると、それを作らなければなりません。

ロボットを作るには、3つの要素技術が必要です。実際に動く部分を作るための機構（メカ）と、それを動かし、またセンサから情報を得るための電気／電子回路、そして動作を決めるコンピュータです。この「機構＋電子回路＋コンピュータ」の複合技術を「メカトロニクス（Mechanics+Electronics=Mechatronics）」*2と呼びます。今回は、その3つの要の1つ、コンピュータ部分をPCとLinuxで作り上げた実例として、写真1aと写真1bに示す2脚歩行ロボットと4脚歩行ロボット*3のシステムを解説します。

【写真1a】脚歩行ロボット（2脚ロボット「Monroe」）



ロボットのハードウェア

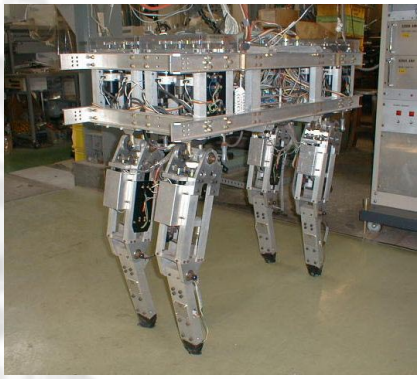
まず、例として取り上げた、2脚歩行ロボット*4と4脚歩行ロボットのハードウェアについて簡単に解説します。

この2台のロボットは、外見こそ違いますが、似た構成となっています。どちらも脚部は12個のモータで駆動していて、ロボットをスリムにするために、クランクスライダ機構*5を使用して関節を動かします。ロボットの股・足首など前後左右方向に曲がる関節は、特殊な機構であるパラレルクランクスライダ機構（図1）によって、2個のモータの協調動作で駆動しています。

これらのロボットには多数のセンサも搭載しています。大きく分けて、モータに直接接続され回転角を測定するもの、ロボットの姿勢を調べるもの、ロボットの足裏／脚先にかかる荷重を測定するもので、制御の際に使用されます。

最近は各種技術の発展に伴って、電源やコンピュータなどすべてを内蔵した自立*6型ロボットが多くなっていますが、このロボットは非自立型で、コンピュータと電源、モータの駆動回路は外に設置しケーブルで接続しています。センサ情報は外部のコンピュータに送り、制御のための演算を行ってモータの出力指令を算出します。これによって駆動回路がモータに電流を流してロボットが動きます。これを1秒間に100回の頻度で（処理によってはより短周期で）ひたすら続けます。

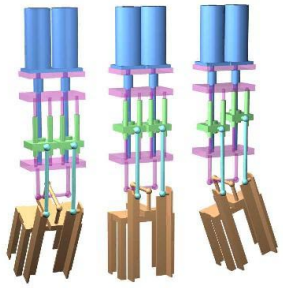
【写真1b】脚歩行ロボット（4脚ロボット）



●2脚歩行ロボット

身長約1.5m、重量約27kgの、人間型の脚を持つロボット（今年で13歳）です。2本の脚を持つロボットには人間に似た構造のものの他に、

【図1】パラレルクランクスライダ機構（2個のスライダが同方向に動くと、関節は前後に（左図）、逆方向に動くと左右（右図）に曲がります）



＊1　制御理論はいろいろとややこしいので、今回はほとんど触れません。歩行ロボットなら、要は転ばずに前進できればいいのですが、これは生やさしいものではありません。歩行ロボットの研究をすると改めて生き物のすごさを思い知らされます。ちなみに、前回にも増して脚注の数が多くなっていますが、大半は本筋にあまり関係のない補足です:-)。
＊2　コンピュータは文字には入ってませんが、必須です。珍しくも、世界標準の和製英語です。46ページのコラム参照。
＊3　これらのロボットは東北大学大学院工学研究科機械電子工学専攻江村研究室の職員、学生（含む卒業生）の手により開発／改良が進められてきたもので、本記事の主語は基本的に研究室です。また、研究の一部は文部省科学研究費補助金によったことをここに記します。
＊4　世の中では2足ロボットという表記を良く見かけますが、我々は「大切なのは足（足首から下：foot）よりも脚（股関節から下：leg）である」との考えに基づいて、「2脚」と呼んでいます。
＊5　自動車のエンジンのようにスライドする運動を回転運動に変換します。モータの回転運動は、いったんボールネジと呼ぶネジ状の部品でスライダの直動運動に変換され、クランクスライダ機構によって関節の回転運動に変換されます。
＊6　外から電源などの供給を受けずに動作することを（脚の有無にかかわらず）自立といいます（←→非自立）。同じ発音ですが、操縦されることなく目的の動作をこなすことができるロボットを自律型といいます（←→リモコンなど）。

竹馬型（足がない）や鳥のようなもの、膝の代わりに伸縮機構を用いたものなどがあります。人間に似た歩行動作を可能とするために、股関節は前後左右に曲がり、鉛直軸回りに回転し、膝関節は前後にのみ、足首関節は前後左右に曲がります。片脚あたり6カ所の回転軸があり（6自由度といいます）、それらを適切に制御することで、ロボットの胴体を基準に、足の位置と向く方向を3次元空間内で自由に決められます（もちろん、可動範囲に限度はあります）。これらは合計12個のモータで駆動しています。これに加えて、爪先には前後方向に曲がり、バネで戻る簡単な関節を取り付けてあります。ちなみに、関節間の長さなどは、身長1.6mの人間の脚とほぼ同じです。

最初は脚だけのロボットだったのですが、腕を振ることの影響を研究するために、2000年、腕が付きました。もともと腕がない設計でしたので、軽量でかつ効果があるように設計されています。図2aに示すように、クランクスライダ機構を2個組み合わせた構造で、1個のモータで図2b、図2cのような人間に近い腕振り動作を可能としました。

制御システムは、3台のPCによって構成されています。1台が全体を統合するマスタコンピュータでPentiumII 400MHzを搭載し、OSはLinuxです。残り2台はモータの制御と、センサの情報処理を分担するスレーブコンピュータで、NECのPC98x1シリーズ（i386 20MHz、およびi486DX 66MHz、OSはMS-DOS）です。このマスタPCの更新が、Linuxで制御をしようと思い立ったきっかけです。スレーブPCも更新計画を進めています。

●4脚歩行ロボット

4脚ロボットの大きさは、長さ940mm、幅430mm、高さ880mm（脚長590mm）で、重量は約50kgです。脚には前後左右に曲がる股関節（と肩にあたる関節）と前後に曲がる膝関節の合計3自由度あり、4本合計で12自由度になります。この3自由度を制御することで、脚先位置を決定でき、4本の脚の動かし方でさまざまな歩行を行うことが可能です。

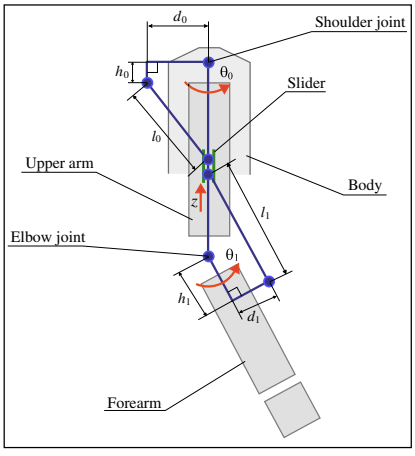
4脚ロボットの制御システムは、一足先に完全にLinuxに移行しました。ソフトウェアについては、2脚ロボットですでに実績のあったシステム（後述）の部品やノウハウを丸ごと移植して簡単に済みましたが、インターフェイス（IF）部分が問題となりました。モータ1個を制御するには、その出力を指令する機能と、回転角を読み取るカウンタを必要とします。モータが12個あるということは、全部で12チャネル必要ということになり、市販のボードではチャネル数が少なく、拡張バスの本数が足りません。そのため、以前よりボードの自作はしばしば行っていたのですが、PCIバスでは簡単にはいきません。手間もかかるので、1枚だけ必要なものをわざわざ作るのは割に合いません。

そこで独自に、写真2に示すようなPCI用の万能IFボードを開発しました。このボードは、PCIとの複雑なやり取りを担当するバスブリッジ（アドテックシステムサイエンス社製APIC21、[3]）と、転送するデータによってさまざまなデジタル回路を構成できるプログラマブルなLSI（ALTERA社製CPLD、[4]、[5]）を搭載したもので、PCIバスに挿して、内部回路を設定して使用するものです。これなら1種類だけ用意しておいて、用途に応じて回路を設計すればよく、使い終ったら他にも転用可能で便利です（安く量産してくださるところ募集中）。4脚ロボットの制御に2枚（1枚あたりモータ8個分）使用しているほか、別分野の信号処理の研究などにも使用しています。

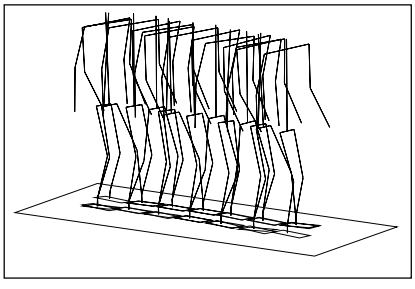
マルチプロセスで制御

そもそも、Linuxでロボットを制御しようと思ったのは、MS-DOSからの移行が目的でした。長いこと、パソコンで制御といったら、NECのPC9801シリーズに拡張ボード*7を差し込んで、古くはN88-BASIC＋アセンブリ言語、その後長い期間にわたってMS-DOS＋C言語が主流でした。ところが、IBM PC互換機の台頭とともに、98シリーズは廃れてきました。モータをいくつか回す程度なら、i486くらいで十分でしたので、しばらくは古いパソコンで粘っていたの

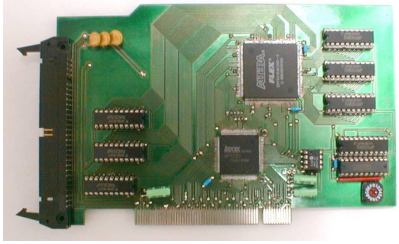
【図2a】2脚ロボットの腕「Monroe Arm」：機構



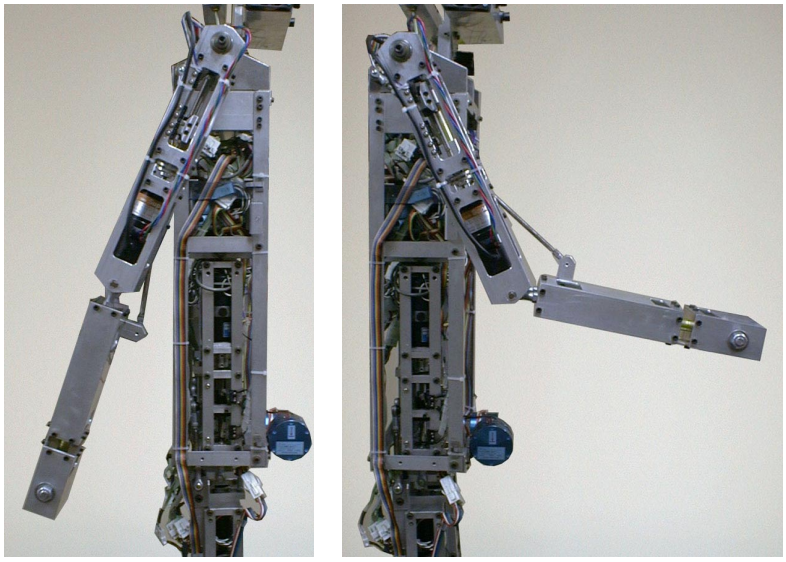
【図2c】2脚ロボットの腕「Monroe Arm」：腕を振る歩行



【写真2】ユニバーサルIFボード



【図2b】2脚ロボットの腕「Monroe Arm」：動作



＊7　98シリーズのCバスは最初から16bitで、しかもアクセスタイミングが速い、ボードがほぼ正方形でバス側と外部側が対向しているので回路のレイアウトがしやすい、箱を開けなくても抜き差しできるなど、ISAバスに対してハードウェア面で大きなアドバンテージがありました。

＊8　制御中に落ちられるのは困りますが:-)、それ以前に中身が不透明です。MS-DOSはプログラムが動き始めたらCPUを占有できるので、制御用としては使いやすいOS（?）です。

ですが、IBM PC互換機を導入せざるを得なくなりました。Windowsは制御用としては不向きでしたので*8、MS-DOSの導入も考えたのですが、すでに32bit全盛時代で、いまさらな感じがありました。このような背景で第1回で述べたようにLinuxを使う方向に走ったわけです。

さて、一定周期で制御演算もできるようになり、ハードウェアへのアクセスもできるようになったので、Linuxでロボット制御ができる目処は立ちました。あとはMS-DOSで動かしていたプログラムをそのまま移植すればロボットは動くはずです。しかしここで、「Linuxはせっかくマルチタスクなのだから、それを有効活用できないか」と考えました。ここでは、マルチプロセスで制御システムを作る利点についてまず述べて、そのあとマルチプロセス制御環境構築のために開発した手法／ツールについてお話しします。

●マルチプロセスの利点

研究としてロボットを動かしているので、たとえカッコ良くとも利点がないならマルチプロセスで動かす必要はありません。そこでまず、利点があるかどうかを検討してみました。

そもそも、ロボットの制御は多くの処理を階層的に積み重ねて行います。モータの回転を制御する部分、センサの情報を処理する部分、座標計算をする部分、そしてロボットの動きを決定する部分などです。これらの部分はそれぞれに独立性が高く、かつ、研究段階ではロボットの動かし方の部分を主に変更し、それ以外は使い回します。そこで、個々の部分をプロセスにしてしまおうという考えです。

このような場合、よく用いられる手法はライブラリです。それぞれをライブラリにして、コンパイル／リンク時に結合します。これに対するマルチプロセス法の利点は、各部の変更の柔軟性です。ロボットはソフトウェアだけ変更するものではなく、ハードウェアに対しても少しずつ手を入れていきます。当然、ハードウェアに密着したソフトウェアはそれに応じて修正していく必要があります。ライブラリの場合、その度にすべての実験用プログラムを再構築する必要がありますが、プロセスの場合、入出力部分さえ気を付けていれば、部品となるプロセスだけ更新すれば良いわけです*9。

次の利点は独立性です。独立プログラムですので、単体で試験可能に製作できます。また、プロセス間通信の部分で切り放すことが可能なので、システムの一部のプロセスをダミーのプロセスと入れ替えておくということもできます。処理を分担するソフトウェアの代わりに人間がデータを与えることも可能です。

3つ目の利点は、処理の実行順序を気にする必要がなくなるということです。MS-DOSのようなシングルタスク環境で多くの処理を行おうとした場合、その順序に頭を悩ませることになります。それほど重くない処理をすべて同じ周期で実行すればいいなら、全部まとめて周期実行させればよいのですが、それに画像処理のような時間を要する処理を組み込もうとすると厄介です。それに対して、マルチプロセスでシステムを構築すると非常に楽になります。まず、処理の単位ごとにプロセスを割り当てます。周期実行が必要な処理なら、本連載第1回で解説した方法で実装しておきます。連続した処理なら、他を気にせず処理だけをするプログラムを作ります。その上で、これらに優先度を指定して実行します。ロボットの動作制御がバラバラの間隔で動作すると困りますので、これを最優先に、画像処理やUIは低めに設定します。すると、Linuxのスケジューラ*10が、周期実行のタイミングになったらロボットの動きを担当するプロセスに切り替え、それが終わったら、余った時間で画像処理をするようにしてくれます。個々のプロセスは、それぞれ単体で目的の動作をするように実装さえすれば、CPUに余力が有り、条件に矛盾がなければ、Linuxがうまく切り替えて実行してくれます。

多少手間は増えますが、このくらいの利点があれば*11、マルチプロセスの制御システムを作る価値があるといえそうです。次から、実際の構築方法を述べていきます。

●プロセス間通信

複数のプロセスで共同作業を行う場合、プロセス間通信が必要になります。これに必要な要件をロボット制御という観点から拾ってみました。

- 即応性
  あるプロセスの出力した情報は、すぐに受け手に伝わらないと困る。
- 新鮮さ
  過去のデータが欲しいこともたまにはあるが、普通は判断には最新データが必要。ストリーム型の通信は、読むのをサボると古いデータが詰まるので、共有メモリ型の形式が望ましい。
- 通知
  マルチタスク環境の基本は「寝て待つ」。上流側のプロセスからの情報到着で、受け側が目を覚まして処理を開始、という動作が必須。
- 情報分岐
  ある情報を生成するプロセスは、常に1つに制限されるべき*12だが、受信は複数のプロセスで自由に可能であることが望ましい。

これらの特性から、パイプやソケットより、共有メモリ型の通信機構が望ましいと考えられます。Linuxには共有メモリ*13が用意されてますが、残念ながら通知機能がありません。いくつかのLinuxの標準機能を組み合わせることで目的は達成できそうですが、それほど面倒でもありませんので、新たにドライバを作ることにしました。

製作したドライバは、第2回でサンプルとして紹介した伝言板ドライバと似た構造のものです（メッセージボードデバイス。MSGBと呼んでいます）。システムコールの`open()`、`close()`、`read()`、`write()`、`select()`に対応した作りで、`read()`、`write()`にて特定領域の先頭から読み書きします。`select()`は領域の更新を待機するもので、待機対象の領域に他プロセスが`write()`すると、`select()`終了になります。領域の選択は前回の例では`ioctl()`を使用しましたが、このドライバの場合は「`"?abcdefgh"`」と、「`'?'`」を識別名に付けて`write()`するようになっています。ドライバ側では該当する識別名の領域があれば、以後それを読み書きするようになり、なければ空き領域に名前を付けて確保します。さらに、ロック機構を追加してあります。これは1個の領域には1個のプロセスのみ書き込めるようにするものです。

この`read()`、`write()`で毎回常に先頭から読み書きという実装は次のデータ形式と組みで効率的に働きます。また、`select()`で待機できるため、ネットワークのソケットと同時に待機したり、GUIをつくる際に、GTK+に委託して一元管理するなどが可能になりました（図3）。

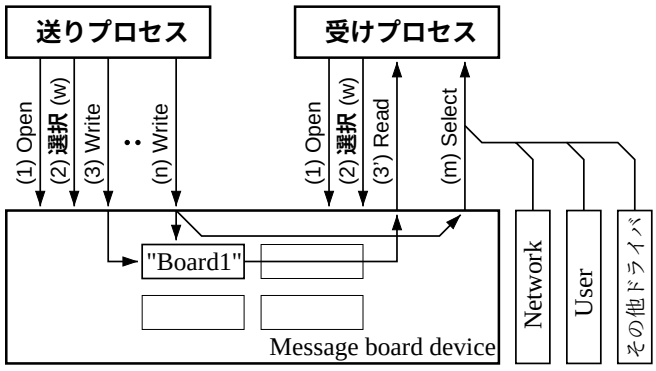
●データ形式VariableSet

MSGBの上では任意のデータのやり取りが可能ですが、利便性を考えて、データ形式を定めることにしました。

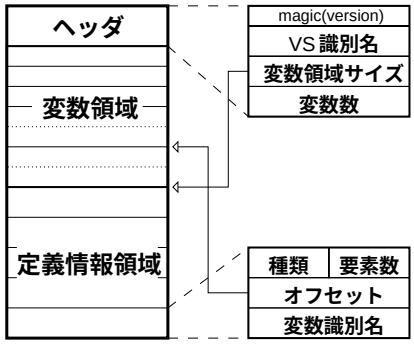
プロセス間でデータをやり取りするために、多くの場合は構造体にデータを詰めて丸投げす

＊9　ダイナミックリンクすればいいのかもしれませんが、変数を増やすなど入出力の形式が変わると面倒なことになりそうです。
＊10　前回、前々回も出てきましたが、実行を要求しているプロセスの中から、実際にCPUで実行すべきプロセスを選定する部分です。
＊11　欠点ももちろんあります。マルチプロセスでシステムを作るための道具立ても必要ですが、何より、制御する度にktermを開きまくって各プロセスを走らせなければなりません:-)。ただ、これは欠点というよりは、利点になることがあります。個々の部分の動作モニタが別々のウィンドウでできますので。
＊12　同一の情報を供給するプロセス2個を誤って同時に動かしてしまうと、短周期で交互に大きく異なる指令がロボットに出される可能性があります。ロボット稼働中にこれが起ると、ハードウェアが損傷する危険性がありますので、避けなければなりません。
＊13　システムコールの`shmget()`、`shmat()`を使用します。

【図3】メッセージボードデバイスの概要



【図4】VariableSetのデータ構造



るという方法がとられます。この方法は作りも簡単で、データ量の点でも効率的なのですが、変数を追加したりすると、結局関連するプログラムすべてを再コンパイルしなければなりません。そこで、構造体を拡張して、VariableSet（以下、VS）と呼ぶデータ形式を制定しました。

VSは構造体にそのデータ構造の定義を添付した形になっています（図4）。先頭にはVSを示す固有のパターン、VSの識別名、データ領域のサイズ、変数の数を記載したヘッダがあり、続いてデータの構造体が続きます。最後に変数の数だけ定義情報が続きます。定義情報には、変数の識別名、種別（整数：long int、実数：double、文字：char）、配列要素数、格納オフセットを記してあります。

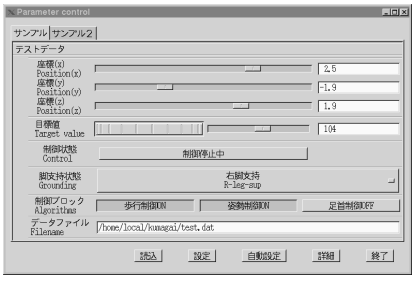
各プロセスでは、次のようにVSを扱います。まず、MSGBに対して必要とするVSの識別名を指定して、ヘッダのサイズだけ領域を読み取り（MSGB用の識別名として、VSの識別名をそのまま使用）、有効性を検証します。

有効であれば、ヘッダの情報から変数領域＋定義情報領域に必要なワークエリアを確保し、定義情報までをすべて読み取ります。これを基に、変数領域のどこに目的とする変数が格納されているかを同定します。そのため、あるプログラムの都合で変数が追加されても、他のプログラムは影響を受けません。

往々にして、データの定義はデータそのものより大きくなりがちですが、VSは1度作ったら定義は変更しないとルールを定めているため*14、2回目以降はヘッダ＋データ領域のみread()、write()します。2回目以降には定義情報にアクセスしないとすることでオーバーヘッドを最低限に押さえました。もし、MSGBから有効なVSが得られない場合は作ることになります。構造体に変数を並べて、その定義情報を作って、ヘッダとともに書き込みますが、この場合も全部書く必要があるのは1回目のみです。

もともと、VSはMSGBの上で使うつもりでしたが、一部デバイスドライバの入出力にもエミュレーション機能を付けました。これによって、次に述べるツール群で一元的に扱うことが可能となりました。

【画面1】VS GUIパネルの一例：生成されるGUIパネル



●VS用ツールの開発

VSを採用した最大の目的は、入出力形式の統一です。その威力を発揮するのは、ここで扱う共通ツール群です。

・VSレコーダ

研究をする上で、データの採取は非常に重要です。これまでは、データを採ろうと思ったら、制御プログラムに大量の配列とfopen()やfprintf()を並べたりしていました。これはプログラムが汚くなる上、気付くとデータが1系列足りなかったり（;_;）、あまり良い実装方法ではありませんでした。それを一挙に解決したのがVSレコーダ（VSR）です。VSRは指定されたVSを監視し、更新され次第記録していくツールです。データはあらかじめメモリ上に確保したリングバッファに順に書いていき、最新の一定期間のデータを常に保持しています。Ctrl＋Cで、その最近のデータをファイルにまとめて書き出します*15。できたファイルはバイナリの塊ですので、gnuplotなどで処理する場合は、VSの識別名と変数の識別名で必要な情報を別途抽出します。

VSRを使うためには、記録したいデータはVSの形で外に出ている必要があります。特に制御用のメインルーチンの内部変数はわざわざ出す必要があり、多少手間ですが、次のGUIパネルで監視するにも使うので一石二鳥です。

・VS GUIパネル

動作状況を見極めるには、非常に多くの状態変数を同時に把握しなければなりません。この場合に、数字の羅列が流れるより、GUIでスライダが動いたりしたほうが把握しやすくなります。また、パラメータの調整も、数値を入力するよりツマミを操作して調整できたほうが楽です。その目的のために開発した汎用UIが「VS GUIパネル」です。

GUIパネルはGTK+をベースに開発した、画面1に示すようなものです。任意のVSを扱うことが可能で、リスト1に示したような定義ファイルを書くだけで、面倒なコード抜きでGUIベースの制御システムが出来上がるという代物です。UI要素としては、全体での比率や粗い調整をするためのスライダ、微調整をする水平ダイヤル*16、整数変数の使い道でありがちなステータス表示、真偽表示、ビットごとのOn／Off表示を可能としています。

・その他おまけ

この他にVSをネットワークで投げる「VSNLink」*17や、Webブラウザで遠隔地から監視するCGIのVSWLinkも試作しました。また、VS汎用ではないのですが、GTK+で画面2に示すような3Dビューワも作りました。これはオンラインでも任意の視点からロボットの状態を表示できますが、主な使い道は保存してあるVSRのファイルを再生して見るためのものです。VSのライブラリには、VSRの記録ファイルをリアルタイムのデータのように見せかける仕掛けを組み込んであり、それによって実現しています*18。図2cはこれによって実際の歩行データから作成したものです。

＊14　何らかの理由で変数の追加をする場合は、該当VSの関連プロセスをすべて停止させ、MSGBの領域の削除を行います。
＊15　リングバッファで最新データという組み合わせは、飛行機のブラックボックスがまさにそうです。事故（動作不良）が起ったときや、実験がうまく行った:-)直後にのみデータ保存をすることで、必要なデータを確保し、不要なデータの保存を避けます。
＊16　スライダの粗さが使い物にならず、サンプルを元に作ってしまいました。10回転するので細かい調整が可能です。ダイアルの縞模様がそれらしく表示されているのが売りです:-)。
＊17　最新情報を片っ端から投げるという用途を考慮し、UDP/IPを使用。
＊18　実はその先があって、ロボットの動作を1コマずつPostscriptファイルに変換できます。それをばらばらにgifにして、アニメーションも作れますが、最大の目的は論文に200ページに及ぶパラパラマンガを付けることでした:-)。

ロボットを歩かせる

さて、いくら道具が揃ったところで、中身がなければロボットは動きません。ここでは、ロボットが動くまでをソフトウェア構成を追いながら簡単に見ていきましょう。

●モータの回転を制御する

ロボットを動かすための基本は、モータの制御です。個々のモータごとに、与えられた目標角度になるように制御した上で、より上位の制御部からモータの角度を指令します*19。

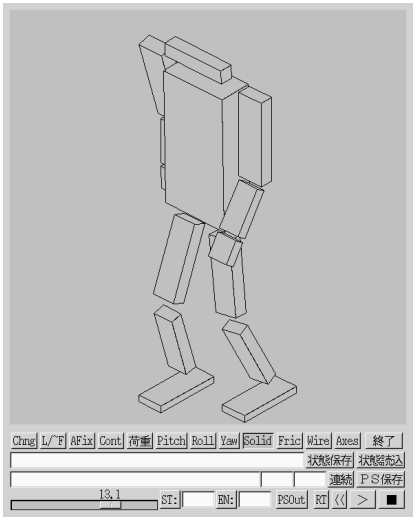
多くのロボットが駆動に使用しているものは、DC（or AC）サーボモータ*20と呼ばれるもので、駆動回路とセットで、指令されたトルク(軸を回す力)や速度で回転するようになっています。さらにこのモータには、ロータリーエンコーダという一定角度ごとにパルスが出る(1000パルス/1回転など)センサや、ポテンションメータという角度を電圧に変換するセンサを取り付けます。前者はカウンタ回路でパルスを数え、後者はアナログ－デジタル（A/D）変換器でコンピュータに取り込み、回転角を測定します。モータの制御とは、この実際の回転角を目標の角度に合わせるように、モータを回すことになります。今回の2脚／4脚ロボットはDCサーボモータ＋エンコーダという組み合わせです。

モータの制御には、一般的に広く用いられているPID制御というものを使いました。Pは比例（Proportional）制御で、目標の角度と現在の角度の差（偏差）に適当な定数を乗じてモータ出力の指令とします（偏差0で出力0）。Dは微分（Differentiation）制御で偏差を時間で微分したもの、つまり実際角度と目標の速度差に定数を乗じて出力とするもので、速度が一致するように働きます。最後にIは積分（Integration）制御で差を時間で積分したものに定数をかけて出力するもので、わずかに残る偏差を解消するものです*21。三者を加えてモータの出力とするために、3つの定数を決める必要がありますが、実験して経験に基づいて決めています。

これを実現するために、ハードウェアとしては、角度を読み込むためのカウンタと指令を出すためにA/D変換器が必要です。ソフトウェア的には、単純な四則演算*22をするだけです。2脚ロボットについては、スレーブコンピュータでCバスに挿した自作IFボードで入出力して、演算はMS-DOSから起動したプログラムで実行しています。角度指令と現在角度などは、市販の共有メモリIFボード経由でマスタコンピュータとやり取りし、マスタ側では専用のデバイスドライバ上でVS形式でアクセスできるように変換しています。2脚ロボットの腕は後で増設したため、マスタコンピュータから直接制御しています。普通のプロセスで制御しても良かったのですが、出来心で作ったデバイスドライバで、位置指令を直接理解する賢いアンプに指令を送っています。これも、VS形式でアクセスします。4脚ロボットはモータの制御もLinux上で行っており、PCIバスに挿したユニバーサルIFボードで入出力し、2ms周期で実行するプロセスで演算をしています。現在値／指令の入出力はMSGB+VS経由で行います。

VS経由にしてあるのは、目標通りにモータが動いているかどうかをときどき検証する必要があるのと、制御プログラムの代わりにGUIパネルをつないで、手動で動かして調子をみることがあるなどの理由のためです。また、この層で複数のプログラムを使い分けることもあります。

【画面2】2脚ロボット3Dビューワ



●足の位置からモータの回転へ

すべてのモータに回転角指令を与えれば、ロボットを自在に動かせるようにはなりましたが、このままでは大変です。足を水平に動かすだけでも、すべてのモータに同時に適切な指令を与えなければなりません。ロボットの制御方法を考える場合にも、脚先や足首の位置、足の向きを動かすことを考えるほうが、個々のモータの動きを考えるよりはるかに楽です。

このような、モータの回転角と足の位置などの間の変換作業を「運動学演算」と呼びます。モータの回転角から足の位置を算出する順運動学と、足の位置からモータの回転角を決定する逆運動学があります。順運動学は、高校の数学で習うようなsin、cosを使った座標の回転を、回転軸の数だけ3次元空間で計算すれば得られます。逆運動学は、簡単なものは直接計算できます*23が、簡単に解けないものは、数値計算でなんとかします。

2脚ロボット／4脚ロボットとも、運動学は2段階に分けて計算するようにしてあります。第1段階はモータと関節角度の変換（関節運動学）、第2段階は関節角度とロボットの胴体を基準とした足の位置や向き、脚先座標の変換(脚運動学)です。分割したのは、両ロボットの股関節、足首関節に使用したパラレルクランクスライダ機構の変換が非常に厄介なためです(数式が解けないので数値計算して作った数表を参照しています)。

実装したプロセスは、順運動学／逆運動学用の足の位置、関節角度、駆動部のスライダ位置(モータの回転角と比例関係)、およびモータの

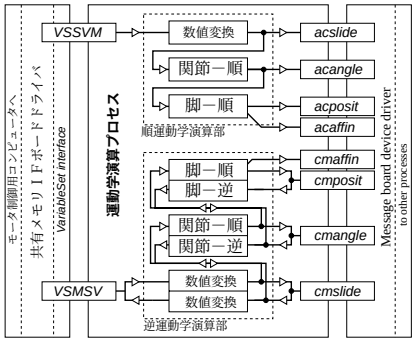
【リスト1】VS GUIパネルの一例：定義ファイル

```
# ページ定義
page testvarc サンプル
# ヘッダ
header テストデータ
parampoint position[3] 0    -5    5  座標(x)\nPosition(x)
                       0    -5    5  座標(y)\nPosition(y)
                       0    -5    5  座標(z)\nPosition(z)
dialint target         0 -1000 1000  目標値 \n Target value
button  system  制御中 制御停止中 制御状態 \nControl
state   foot    4 支持無し \nfloating 右脚支持 \nR-leg-sup
    左脚支持 \nL-leg-sup 両脚支持 \nDubble-sup 脚支持状態 \nGrounding
buttonset cont  3 足首制御 ON 足首制御 OFF 姿勢制御 ON 姿勢制御 OFF
    歩行制御 ON 歩行制御 OFF 制御ブロック \nAlgorithms
string  filename[200]  データファイル \nFilename

page testvarc サンプル2
```

＊19　複数のモータの面倒をまとめてみる制御方法もあります。
＊20　近年ロボット作りにポピュラーになってきたラジコン用サーボとは別物です。
＊21　最後に摩擦力などでPでは動かないような微妙な偏差が残ったときに、その差を徐々に積分していき、たまったところでモータが摩擦を振り切って動いて目標に近づく、とイメージすれば分かりやすいでしょうか。
＊22　微分の代わりに、前回の偏差と今回の偏差の差をとって演算の時間間隔で割ります。積分の代わりに、現在の偏差に時間間隔を乗じて、積分結果の変数に加算（+=）します。
＊23　例えば、股関節と足首関節の距離から、膝関節の角度は直接求まります。その他複雑なものでも直接計算できる場合があります。

【図5】運動学演算プロセスの構成



回転角度のVS入出力を持つものとしました。2脚用のプロセスの詳細を図5に示します。図の右側の「ac」で始まるVSは現在値を、「cm」で始まるVSは指令値を示し、それぞれ位置（posit）、関節角度（angle）、スライダ（slide）からなります。affinは、座標変換を簡単にするための行列群です（3Dビューワはこれを使用）。順運動学側はロボットのモータの実際の回転角をスレイブからVSSVM経由で得て、ロボットの姿勢を算出し、制御での使用や動作記録を可能にします。逆運動学側は、足の位置でも、関節の角度でも指令できるようにして、演算結果はモータの回転制御側（VSMSV）に出力します。主たる方向は逆運動学演算ですが、内部には順運動学計算も含めてあります。これは複数の指令入力を連動させるためのもので、例えば関節角度で指令した場合、順運動学演算も行ってcmpositに出力します。こうすることで監視を容易にするとともに、ロック機構を有効にして、他のプロセスが別の指令VSに書き込んで指揮系統が乱れるのを防止します。

この運動学のプロセスはロボットの制御プロセスと密接な関係を持つものです。運動学演算プロセスは常時指令入力のVSの更新をselect()で休眠待機した状態にあります。ロボットの制御プロセスが足の位置を算出して、足の位置指令用のVSに書き込むと、運動学演算プロセスが目を覚まし、すぐに変換作業を開始する、という連携作業が行われる設計になっています。本来、両者を1つに結合してもいいようなものなのですが、あえて分割しました。部品化することで脚の機構を手直ししたときに、運動学演算プロセスだけ修正すればよくなりますし、指令VSをGUIパネルで手動操作して動作試験もできます*24。また、データの記録を可能にする意味もあります。

●センサの情報を処理する

モータのところでも角度を測定するセンサが出てきましたが、それとは別に、ロボットには姿勢の傾きを測定する姿勢センサと、接地荷重を測定する力センサが取り付けられています。実のところ、関節を指定通りに回転できれば、ある程度は歩行できます。しかし、姿勢センサなどを用いない場合は、路面に少しの傾きがあってロボットの姿勢に影響が出た場合などに、補正ができず倒れてしまうことがあります*25。

2脚ロボット、4脚ロボットとも、胴体に3種類計7個の姿勢センサ*26を持っていて、胴体の前後方向、左右方向の傾斜角と鉛直軸回りの旋回角度を測定できます。また、2脚は足の裏に計12個、4脚は各脚先に1個、計4個の力センサを取り付けていて、ロボットの重心位置*27を測定できるようになっています。

2脚ロボットでは、モータの制御同様、別のPCに委託してあり、算出済の姿勢角、重心位置を共有メモリボード→VS経由で得ます。4脚ロボットでは自前で、PCIバスに挿したA/D変換ボードでセンサの値を定期的に読み取り、姿勢センサ信号の合成処理などを行います。これも周期実行プロセスで、5ms程度の周期にしています（A/Dボードが賢く、センサの値自体は1msで取得）。結果はVSとしてまとめて出力し、ロボットの制御に用いたり、歩行状況の記録をするなどします。

●ロボットを動かす

ロボットを歩かせるための部品は揃いましたので、あとは脚の動かし方を指令するのみです。

制御用のプロセスでは、センサ処理プロセスが取得したセンサ情報や、運動学演算プロセスが算出した現在の足位置をもとに、どう動かすかを考えて、足の位置を計算し、指令することになります。この指令は、滑らかに動かすために、ある程度短い周期で定期的に連続して出力する必要があります。そのため、主制御プロセスも周期実行します。指令は運動学演算プロセスに伝わり、それがモータ制御の目標値になります。

歩行の研究では、歩行の周期を変えたり、歩幅を変えたり、試行錯誤して歩き方を見付けることもあります。そのためのパラメータ類もVSで外から受け入れ、内部で作った中間情報も出力するようにしています。右手はマウスを握り、左手はロボットを支えて実験していました。

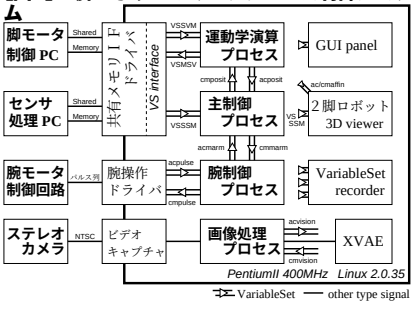
このような要素を組み上げて、これらのロボットは動かされています。すべて組み上げると図6に示すようになります。実験内容に応じて画像処理をつないだり、必要な部品を組み合わせて動作させています。徹底的なまでにVS仕様にしたため、モニタしたり手動操作に切り替えるのが容易になりました。操作は画面3のような画面で行います。右上の仮想デスクトップの表示が、マルチプロセスぶりを物語っています:-)。

ロボットに目を付ける

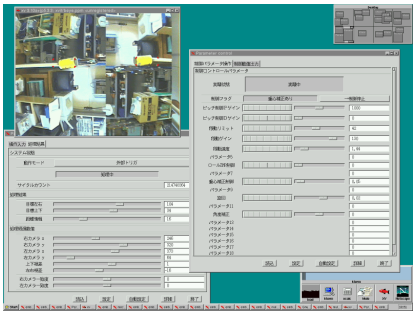
写真1aでも確認できますが、2脚ロボットには目が付いています。これは光ったりする飾りではなく、周囲の状況を取り込むための2個の小型のCCDカラーカメラです（4脚には取り付け計画中）。

カメラの小型軽量化により、ロボットにカメラを搭載することが多くなってきました。しかし、ロボットにカメラを付けるだけでは役には立たず、得た画像を処理して、情報を抽出して、初めて有効なセンサとなります。口で言うのはやさしいのですが、最先端の研究でも、人間並みに周囲の状況を判断することはできません。コンピュータの著しい性能向上や、専用のハードウェアによる支援で処理速度は劇的に向上しており、教えられた物が映っているかどうかくらいは比較的短時間で判別できるようになりました。しかし、映っている物を列挙せよと命じても、物と物の境界を探すための万能の方法すら確立されていません*28。とはいえ、使い方を工夫すれば、便利なセンサになることは確かで、ロボットの制御にも使って効果を上げています。

【図6】2脚ロボットのマルチプロセス制御システム



【画面3】2脚ロボットの制御画面の例



＊24　運動学演算プロセスが出来て、最初にやったことは気合いと根性の手動歩行でした:-)。足首の位置、足の向きで両足で12個の数値をロボットが倒れないようにそろそろ動かして2歩歩かせることに成功しました。
＊25　特に人間サイズの2脚歩行ロボットは、倒れたときが最後です。全身打撲の複雑骨折で復活困難になります（合掌）。なので、普通はワイヤなどで完全には転ばないようにゆるく吊って実験します。
＊26　カーナビやビデオカメラの手振れ補正に使用されるレートジャイロという角速度（倒れる速度）を測定するセンサと、傾斜計という振り子のような重力方向を測定するセンサを装備してます。3種類も複合して持っているのは、それぞれのセンサに得意不得意があるためで、合成して1つの角度情報にします。
＊27　静的な重心位置は各部の重量から計算できます。ここでは勢い（慣性力）も含めた動的な重心位置を計測します。
＊28　もちろん、同じものが沢山あるとか、色が違うとか、そういった制約条件を付けたものなら、種々の方法が研究されています。さまざまな状況に対処できる、人間並みの判別能力には、まだ追い付きません。

●ステレオビジョン

ロボットの頭には2個のカメラが取り付けられています。人間が片目で見るより両目で見た方が奥行き感覚が得られるように、2個のカメラの画像を処理すると、奥行き方向の情報が得られるためです(人間の場合は知識と照合して、片目でもある程度なんとかなりますが)。このように2個のカメラを使った視覚センサを「ステレオビジョン」と一般に呼んでいます。

原理的には、異なる2カ所から対象を観察したとき、見える方向に差が生じることで、三角形を定義する条件の1つの「一辺とそれを挟む二つの頂点の角度」が得られ、もう1つの頂点に位置する対象物の位置が確定する、というものです。人間の場合は対象を両目で注視して、そのときの眼球の向きから絶対的な距離感を得て、目に映った像の上でのずれから相対的な前後位置を判断します。それを模して、カメラを動くようにしたロボットもありますが、今回は処理を単純にするために、カメラは固定としました。

●ステレオカメラ

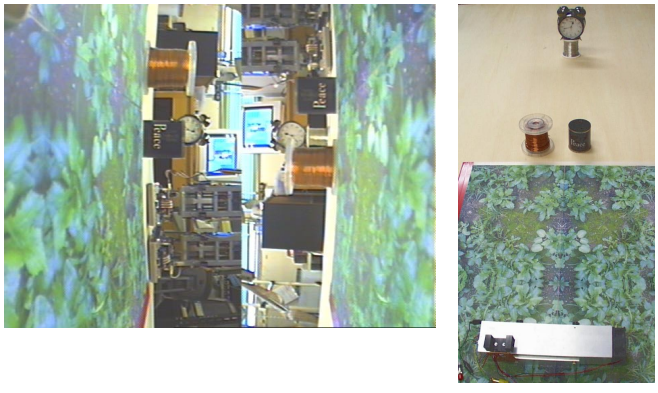
さて、カメラはただ2個付ければいいわけではありません。どのような目的で使うか、どのくらいの範囲を見る必要があるかなどをもとに選定し、取り付ける必要があります。現在使用しているカメラは、多少距離があるところを、広い範囲で見るようにしたため、逆に至近距離や細かなところは見えない仕様になっています。

加えて、2個のカメラは同期するようにしました。詳しいことは省きますが、カメラの信号が完全に同期していると、映像信号を切り替えても、ビデオキャプチャカードの映像が乱れません(非同期の場合は切り替えると短時間ですが乱れます)。また、簡単な電子回路で画像を1画面に合成することもできます。さらに、動いているものをステレオビジョンで処理する場合(あるいは、カメラの方が動く場合)、2個のカメラで同時に撮影しないと、正確な計算ができません*29。そのため、同期させることが好ましいのですが、カラーで小型で安くて同期機能のあるカメラは、残念ながら見付かりませんでした。そこで、秋葉原で1個1万円ちょっとで買ってきたカメラを2個改造して、同期させて使っています(安さに自信あり:-))。

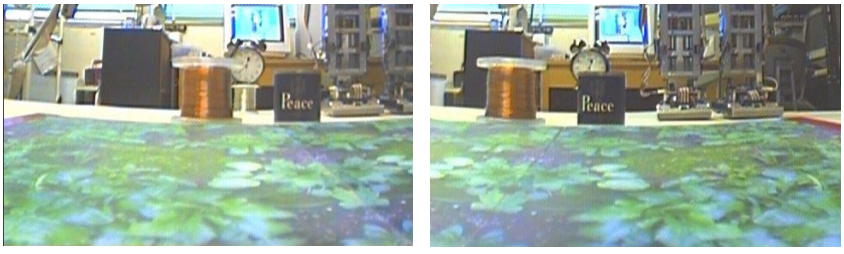
画像の取り込みには、市販の画像キャプチャカードを使っています。Linuxのドライバ付きと銘打った専用品を使っていたのですが、最近Video for Linuxの使い方が分かったので(インターネット上で検索するといくつか親切な解説があります)、Brooktree社製Bt878チップを載せていることだけを確認して安いキャプチャカードを買ってきています*30。

取り込んだ画像の例を写真3aに示します。配線を減らして、取り込みが1回で済むように画像を合成しています。写真3aの右の画像は、写真3aの左の画像から分かりやすいように対応領域を切り抜きました。もちろん、コンピュータ内部では、分割せずに一括して処理をしています。

【写真3a】ステレオカメラによる撮像:取得画像例と物の配置



【写真3b】ステレオカメラによる撮像:左右のカメラの画像に分解



●画像処理ソフトウェア

最初にも述べましたように、画像がとれるだけでは役に立たず、情報を得るには処理をしなければなりません。現在までに著者が2脚ロボットでカメラを活用した例は、

- 教えられたものを追尾して歩くために目標を検出し、その方向と距離を求める。
- 斜面を歩くために、地面を見てその斜度を推定する。

です。このロボットは、足の裏の摩擦の関係などで、真っ直ぐ歩かせているつもりでも徐々に曲がってきます。そこで、ロボットの前に目標をおいて、歩く方向をコントロールしようというのが1つ目です(人間も目をつぶったら曲がりますが、目標を見ていると真っ直ぐ歩けます)。2つ目は難しそうですが、基本原理は単純です。坂道を歩くとき、登り坂では地面が近く見え、下り坂では遠く見えます。この関係を応用し、カメラで斜め下を一定角度で見て、地面までの距離を測って斜度を推定します*31。

目標を発見し、距離を得るためには、テンプレートマッチングと呼ばれる手法を使っています。これは、テンプレートと呼ぶ小画像と、得た画像の各所を比較し、なるべく一致度が高い場所を探し出す、という手法です。目標を見付けるには、あらかじめ目標の画像をテンプレートとして用意しておき、取得した画像から探し出します。見付けた場合、本来目標が見えるべき画面内の位置と、見付かった位置の差から、目標の方向が分かります。距離を求めるには、両目の画像内で目標を見付けます。このとき、目標が遠方にある場合に比較し、目標が近くなるほど、両目で見た目標の位置のずれ(視差)が大きくなります(写真3bを参照)。この視差を距離の情報として使います。地面までの距離を求めるには、右目の画像の地面らしきところを切り抜いてテンプレートにして、左目で探せば、やはり視差が分かります。

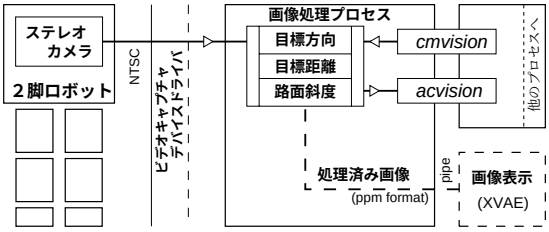
この処理の基本となっているテンプレートマッチングという処理は膨大な演算量を必要とします。まず、1カ所を評価するために、テンプレートの画素数に比例した回数の演算が必要です。これを画面全域でやるとテンプレート画素数×画面の画素数になります(テンプレート32×32ドット、画面640ピクセル×480ピクセルで約3億回、カラーでその3倍)。そのため、演算1回の処理を高速化すると処理時間の短縮に絶大な効果があります(これは画像処理全般にいえることです)。そこで、このテンプレートマッチングの中枢部分はPentium MMX命令を用いて実装することにしました。複雑な演算の

＊29　本来、両目で見えている対象の位置の差は、対象までの距離のみに依存するはずですが、動いているものを異なるタイミングで撮影しても、時間差×速度だけ差が出てしまいます。
＊30　今のところ3枚買ってハズレなし。2枚挿しもOK。付いてきたリモコンがもったいないので、それも使えるようにしてみたり:-)。制御用のPCで実はテレビを見られるということは、皆には秘密:-P。
＊31　東北大学の工学部は山(?)の上にあるのですが、麓のキャンパスに遊びに行った帰りの山道(?)でこの方法を思い付きました。

すべてをアセンブリ言語で書くことは容易ではありませんが、数をこなさなければならない単純計算の部分はアセンブリ言語でチューニングすると効果があります*32。さらに、MMX命令は複数のデータを同時に（1byte×8、2×4、4×2）処理可能なので、画像処理にはもってこいです。今回はデータの配列とも相談して、赤、緑、青、輝度の4成分の計算を同時に進めるように実装しました。出てきた4個の評価値は、テンプレートの各色ごとの複雑さを加味して、加算して使用することでカラー情報に対応しました。同じ内容をC言語で最適化オプション付きでコンパイルした場合に比較し、数倍の速度向上が得られています。

ソフトウェア部分の構成を図7に示します。上に述べたように、処理の中核をMMXで記述した画像処理プロセスが、画像を取得し、目標の情報や斜度を算出して、VSで出力します。処理結果は数値で出てくるのですが、それだけでは何を見ているのか、目標を正確に見付けているのかを判断することは困難ですので、見付けた目標に印を付けるなどした画像をJohn Bradley氏作のグラフィックビューワ「XV」（[6]）に送ってモニタすることを可能にしています。XV自体には、他のプロセスから連続して画像を受け取る機能はありませんので、それを拡張した拙作の「XV Algorithm Extension」（以下、XVAE）＊33を使用しています。XVAEは画像処理を手軽に行えるように拡張したもので、pnm形式で入出力可能な処理プログラムを用意すれば、メニューに登録して処理を実行できるようになります（プラグイン的動作）。入出力両方ともサポートしている必要はなく、ppm形式で画像を出力するキャプチャプログラムを作れば、XVにビデオキャプチャ機能が付きます。また、多段のundoや、1度やった処理手順を別の画像に連続して適用する機能などがあります。画面3の左上にあるのがXVAEで目標の空き缶:-）にロックオンしています。

【図7】画像処理システム



●ロボットの動作への統合

目標の情報や斜度の情報が得られましたので、あとは歩行に反映させるのみです。「のみです」とはいっても、そこが面倒なところで、当然斜度の推定には誤差やミスがありますし、目標が見付からないこともあります。

まず、目標を追尾するためには、距離と方向に合わせて歩き方を調整します。方向がずれていたら、ロボットの前進とともに足の向きを変えることで曲線歩行にして合わせます*34。目標までの距離は1歩ごとに監視していて、近付き過ぎたら立ち止まり、ある程度離れたら歩き始めるようにしました。斜度については、斜面の角度に合わせて歩き方を変更するようにしてあります。これで、起伏地形を模擬した環境での、斜度を推定しながらの追尾型自律歩行に成功しています。

おわりに

今回は最終回ということで、技術情報というよりは技術解説的読み物としてまとめてみました。世の中にはさまざまなロボットがあり、さまざまなシステムで動いています。今回紹介したものは、その中の1つに過ぎませんが、なんとなくでも、ロボットがどのような処理をしながら歩いているかが伝われば、と思います。

ロボットは、メカ／電子回路／コンピュータの複合技術に成り立ってはいますが、最終的な動作を決定する部分には理論があり、理論に基づいたアルゴリズムがあり、それを実装したプログラムがあります。ハードウェアの用意さえできれば、ソフトウェアの世界ですので、本誌の読者の方なら、特に身構える必要のない分野だと思います。本記事では、その境目にある、いくつかの有用なノウハウの公開を目標として執筆しました。ハードウェアの世界に足を踏み入れるための参考になれば幸いです。

Column 機械の進化

長い間、機械は機構のみですべてをこなしていました。動力源は1つで、それを減速したり、回転から直線運動に向きを変えたりして動かしていました。そこに電子回路が加わり、電子制御技術が発展してきました。機構に複数のモータなどを取り付け、それらを電子回路で順に動かしたり、センサ情報を反映させて動かすようになったのです。機構のみでは難しかった仕事もこなせるようになってきました。そして登場したのがコンピュータです。電子回路のみで制御しているうちは、動作を変更するには回路を変更しなければならず、まだまだ不便だったものが、プログラムを変えるだけで済むようになりました。また、より複雑な条件判断もできるようになり、機械の機能が飛躍的に向上しました。最初は大型のミニ（？）コンで制御が試み始められ、すぐにマイクロプロセッサが登場し、急速に新技術として広がりました。

このような進化は身の回りでよく見られます。ミシンは昔は複雑なメカニズムで模様縫いなどをこなしていましたが、今はコンピュータ制御でさらに複雑な模様を簡単に仕上げます。自動車のエンジンも、少し前までは純メカでしたが、現在はコンピュータ制御のものが見られ、燃費の向上などに一役買っています。もちろん、PC内部のディスクドライブなどの可動部も、このような技術の上に成り立っています。

（熊谷正朗）

Resource

[1] Linuxでロボットを作る「その1　Linuxによるハードウェア制御の基礎」
熊谷正朗／LinuxJapan 2001年7月号 40ページ

[2] Linuxでロボットを作る「その2　ハードウェア操作のためのデバイスドライバ」
熊谷正朗／Linux Japan 2001年8月号 58ページ

[3] アドテックシステムサイエンス社「APIC21」
http://www.adtek.co.jp/seihin/apic/apic21.html

[4] ALTERA社
http://www.altera.com/

[5] 株式会社アルティマ
http://www.altimanet.com/

[6] XV公式Webサイト
http://www.trilon.com/xv/
John Bradley氏の手によるグラフィックビューワーのWebサイト

＊32　プログラム自体はまずC言語で書きます。次にアセンブリ言語に落しやすいようにアルゴリズムを変更し、内側のループから順にインラインアセンブルを使って置き換えていきます。
＊33　XVAE化のパッチは、しばらく前から当方のWebサーバで公開しています。jp-extensionパッチを当てることが前提なので、一部パッチとは相性が良くありません。
＊34　5年くらい前から曲線歩行はときどきやっていたのですが、世の中の情勢によると珍しいらしいです:-)。無論、歩行速度次第ではありますが。