2021-05-30 誤植あり版（一箇所）

シェルスクリプトを用いたUNIX哲学に基づくリアルタイム制御

柳戸 新一
（有）USP研究所
s.yanagido@gmail.com

松浦 智之
（有）USP研究所
richmikan@richlab.org

鈴木 裕信
（有）USP研究所
h-suzuki@usp-lab.com

大野 浩之
金沢大学
hohno@staff.kanazawa-u.ac.jp

要旨

ソフトウェアを中心としたシステムの移植性・持続性は重要な観点である．*Gancarz*の*UNIX*哲学においてはハードウェアに依存する高性能なシステムより，多少性能が劣っても移植性に優れたシステムの方が将来的に淘汰されにくいとされる．筆者らはこの*UNIX*哲学に基づき，移植性・持続性を最大まで高めるために，シェルスクリプトのストリーム指向を活用した開発手法を提案してきた．これらの手法はサーバやものづくりをはじめとする幅広い分野で適用され，その有効性が実証されてきた．

一方，これまでリアルタイム制御を必要とする組込み型システムで*GNU/Linux*を用いる場合，低遅延カーネルを利用したり，スケジューラのスケジューリングポリシーをリアルタイム（*RT*）クラスに変更するなどの工夫が行われてきた．しかし，スケジューラの変更はマルチタスク処理を活かしたストリーム指向プログラミングと相性が悪く，並列処理が実施されなくなるといった危険性がある．

本研究ではリアルタイム制御を行うシステムに対しても*UNIX*哲学を適用可能であることを示すために，標準スケジューラで動作するシェルスクリプトを用いた倒立振子の安定化を試み，評価を行った．検証の結果，本手法に一定の実用性があることが実証された．

1. はじめに

操作対象となるシステムを電子回路を用いて計測ならびに制御を行う場合，コンピュータを使ったデジタル処理が広く行われている．このコンピュータにOSを搭載し，OSにUNIX系OSを用いることにより，次のようなこれまでにない利点が得られる．まず豊富なUNIXコマンドを用いたデータの利活用が挙げられる．また，多くのUNIX系OSは既にTCP/IPスタックを持っているので，インターネットと連携するIoT機器の製作も容易に可能となる．そして筆者らが考える最大の利点は，数十年以上に渡って積み上げられてきたUNIXの高互換性・持続性により，UNIX上で制御機構を実装すれば，そのシステムやソフトウェア資産の寿命が何十年に渡ると期待できる点である．

しかし，UNIX環境ではタスクスケジューラの影響でプロセスの実行タイミングが正確に予測できず，リアルタイム制御に支障をきたしてしまう．これまでリアルタイム制御を必要とする組込み型システムでGNU/Linuxを利用する場合，実行時のプロセスのプライオリティ値を変更したり，あるいはスケジューラのリアルタイム・ポリシーをSCHED_FIFOに設定するなどしてプロセスの優先度を高めるといった手法を用いられてきた[4]．また，それに加えて利用するLinuxカーネルを低遅延カーネルに置き換えるなどの工夫が必要となる[5]．このようにプロセスに優先度を設定する場合，どれだけの優先度を与えれば良いか，また優先度を与えたときに，他のプロセスにどのような影響を与えるかを考慮に入れなくてはならない．

そこで本論文では，フィルタや制御量計算といった自動制御で用いられるデータの連続的な変換を，UNIX が持つパイプラインの仕組みで実現する手法を提案する．このとき，スケジューラはディストリビューションの標準である Completely Fair Scheduler（CFS）の設定のまま使い，優先度をあげるといった処理は行わないことを前提としている．これは，Gancarz の UNIX 哲学 [3]（以下，UNIX 哲学）に基づいたリアルタイム制御のための手法であり，以降で本手法の有効性を検証する．

Gancarz は UNIX の考え方として，9 個の定理と，10 個の小定理を示している．その中で本論文の開発手法が従う定理と，その定理を満たすために採用した手法との対応ついて次に示す．

定理 1 (小さいものは美しい).
対応 : すべての処理を行う一つのプログラムを作成するのではなく，機能ごとにプログラムを作成.

定理 2 (一つのプログラムには一つのことをうまくやらせる).
対応 : 適材適所の考え方を導入した，ものグラミング 2[12] に基づいたハードウェア構成.

定理 4 (効率より移植性を優先する).
対応 : 移植性の高い開発手法である POSIX 中心主義 [18] に基づき，加えてスケジューラは変更を加えない標準のものを使用.

定理 5 (数値データは ASCII フラットファイルに保存する).
対応 : プログラム中のどこに流れているデータであっても即座に保存できるよう，プログラム間のやり取りは数値データであっても ASCII データを利用.

定理 7 (シェルスクリプトによって梃子の効果と移植性を高める).
対応 : 各プログラムはコマンドの形式を取り，シェルスクリプトを用いて協調して動作.

定理 9 (すべてのプログラムをフィルタとして設計する).
対応:シェルスクリプトのストリーム指向を活用した設計.

小定理 6 (同時に考える).
対応 : シェルスクリプトのストリーム指向を活用した並列処理.

2. 関連研究

まず，筆者らが提案してきた UNIX 哲学に基づく手法について述べる．松浦らはプログラムの移植性向上を目的とした標準化規格である POSIX.1(IEEE Std 1003.1) 文書に記載された仕様に極力準拠しつつ，さらなる互換性向上のためのプログラミング手法，POSIX 中心主義を提唱し，その有効性を実証してきた [18]．また，大野らはこの手法をものづくりに応用し，POSIX に準拠した Raspbian（現 Raspberry Pi OS）を搭載可能な Raspberry Pi[6] に加え，必要に応じてセンサ・アクチュエータとのインターフェースとなる Arduino[1] を使用した，ものグラミング 1[13] ／ものグラミング 2[12] を提唱し，普及活動を行っている．

また，本論文での報告内容は筆者らの一連の研究の一部であり，試行錯誤によって得られた他の結果については松浦らの報告 [19] を参照されたい．

Linux をリアルタイム処理で活用する試みは数多く行われており，先に述べたプロセスの優先度変更や低遅延カーネルでの利用の他にも，複数の手法が存在する．リアルタイム処理を行うための Linux カーネルとしては RTLinux[10] や ART-Linux[11] が知られている．また，清水らは 2010 年にカーネルの変更に加えて POSIX の範囲内でリアルタイム処理を行うための枠組みを提案している [16].

熊谷らは 2004 年に通常の Linux（カーネルバージョン 2.4 以前）を用いたリアルタイム制御と RTLinux によるリアルタイム制御の違いついて述べ，標準の Linux を用いる利点としては開発の容易さと互換性の確保を挙げている [14]．また，標準の Linux 環境における周期的なプロセスの実行のために，`usleep` 関数を用いる方法を提案しており，RTLinux を用いなければならないほどの精度でなければ `usleep` 関数でほぼ周期的な処理が可能な場合もあることを示している．

3. 提案手法

3.1. シェルスクリプトのストリーム指向を利用した制御

UNIX 哲学で挙げられている定理を満たすために，シェルスクリプトを用いたシステム制御を提案する．シェルスクリプトで推奨されるストリーム指向のプログラミング手法は多くの利点を持つ．その利点は次の通りである．

- シェルスクリプトおよびそれに使用するためのコマンドは，持続性･移植性が高いといわれる POSIX[7] の範囲で記述可能である．これにより将来的なシステムのアップデート等で動かなくなるというリスクを，最小限に抑えられる．

- 容易に変更･拡張が可能である．シェルスクリプトはコマンドをパイプで連結して記述する．このことから各コマンドの変更を行っても他のコマンドの動作に影響を与えることは少なく，新たな制御アルゴリズムの導入もコマンドの置き換えで済ませられる．

- 容易にデバッグ，ロギングができる．任意のコマンド間に tee コマンドを挟むことによってそこに流れるデータをログとして保存できる．加えて，ストリーム指向で記述することにより，パイプを延長するようにプログラミングができ，流れるデータを確認しながら開発を進められる．

- セキュリティホールとなる危険性が比較的少ない．UNIX 環境という下地の上で動作するので，シェルスクリプトそれ自体がセキュリティホールとなる可能性が少ない．UNIX 環境は一般に広く使用されており，セキュリティ強度はそれに準ずる．

また，シェルスクリプトのストリーム指向は，制御工学におけるブロック線図と相性がよい．これはどちらもデータの流れを記述するという点で共通であるためである．ブロック線図における伝達要素，分岐点，加え合わせ点はシェルスクリプトにおけるコマンドに対応付けることができ，信号線はパイプに相当する．

図 1 は本システムで扱うフィードバック系のブロック線図である．図中の破線はシェルスクリプトでの対応付けを示している．制御対象 G はシリアルインターフェースで接続されることを想定し，入出力には cu コマンドを介する．cu コマンドは 1980 年代初頭の UUCP 時代に開発されたコマンドの一つで，UNIX シェルからシリアルインターフェースを操作する機能を持つ．本研究ではこの cu コマンドをシリアルインターフェースと標準入力の相互接続に利用した．分岐点は tee コマンドを使用し，一方の観測値ベクトル $\boldsymbol{y}$ をログとしてファイルに落とし，他方に名前付きパイプ named_pipe を用いることでフィードバックを実現する．制御器 controller は加え合わせ点と C を包含させている．これは PID 制御のような目標値 r が必要な制御では，r をその制御コマンドのパラメータとして設定するためである．controller では内部で偏差 $\boldsymbol{e}$ を計算し，それを $\mathbf{0}$ に近づけるよう制御を行う．得られるセンサ値を $\mathbf{0}$ に近づけるような制御の場合は，この r は与えない，もしくは 0 を与えるという方法を取る．また controller の入力に制御量 u が必要な場合，簡単化のために u は $\boldsymbol{y}$ に含めることにする．

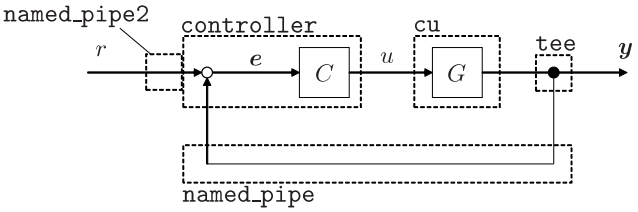

図 1. 本システムのブロック線図とコマンドの対応

図 1 の構造をシェルスクリプトのワンライナーで表現すると次のようになる．

```
$ cat named_pipe | controller -f named_pipe2 |
 cu -l /dev/G | tee named_pipe > y.output
```

名前付きパイプ named_pipe によってデータの循環が表現できていることがわかる．名前付きパイプを使用するためには，mkfifo コマンドを用いてあらかじめ名前付きパイプを作成しなくてはならない．また実際には，controller は制御用のコマンドに置き換える．加えて，データの変形や列の入れ替えのために，コマンド間に別のコマンドを適宜挿入する．

シェルスクリプトでリアルタイム処理を行うときに問題となるのが，各コマンドのバッファリングである．バッファリングはコマンドを高速に動作させるために実装されている機能であり，これによってある程度出力データが溜まるまで，データは出力されずに内部で保持される．しかし，リアルタイム処理ではデータを処理する時間に意味があるため，このバッファリングがリアルタイム制御の妨げとなってしまう．C 言語をはじめとする多くの言語の標準入出力では，コマンドの標準出力が端末ではなくパイプに接続されると，標準でフルバッファリングが有効になる．これはプログラム内部で設定を変更すれば無効化できるが，既存のコマンドを使用する場合，このような方法を取ることができない．そのため，フルバッファリングをコマンドの外部から無効化する方法が必要となる．

フルバッファリングを無効化するために，今回は ptw

コマンド [9] を用いた．バッファリングを行う例として，awk コマンドや Python スクリプトが挙げられる．awk の実装の一つである gawk にはバッファ内のデータを出力するための関数 fflush がある．しかし mawk など，他の awk 実装では使用できないという問題がある．バッファリングを無効化するために stdbuf コマンドを使用することも可能だが，Python スクリプトは内部で独自にバッファを管理しており，stdbuf も汎用的な手段ではない．一方，ptw コマンドは対象となるコマンドに，パイプではなく端末が接続されていると判断させ，フルバッファリングを無効化させようとする．このやり方は汎用的であり，バッファリングを無効化させたいコマンドを ptw でラッピングさせるだけと使用も簡単であるため，この方法を用いた．

この一連のスクリプトは，ディストリビューションの標準スケジューラを用いて動作させるが，一般には組込みのようなリアルタイム性が重視される場合には専用のスケジューラが使用されることが多い．例えば，低遅延カーネルを選択する，プロセススケジューリングをリアルタイムに設定する，あるいはプライオリティを優先するなどの手法がそれにあたる．しかし，プロセスに優先度をつけることは，優先度の低いプロセスが処理されない事態が発生する可能性がある．今までこのリアルタイム性とのトレードオフは，組込み Linux の一つのリスクであった．

本研究では実験時点で最新の Linux カーネル 4.19.66 のデフォルトスケジューラである CFS を用いる．CFS は，全てのプロセスが同じ時間だけリソースを取得するように設計されたスケジューラである．CFS は Linux カーネルバージョン 2.6.23 から使用されるようになったスケジューラであり，設定を変更しない場合には標準でこれが使われる．特別なスケジューリングを行わずに制御が可能であるならば，前述のリスクを負わなくていいという点で優れた手法である．

3.2. 倒立振子の安定化による検証

上記のような手法でリアルタイム制御が可能となるのかという検証のために，今回は本手法を用いた同軸 2 輪車型倒立振子の安定化を試みた．

倒立振子とは重心が支点よりも高いところに位置する振り子のことをいう．同軸 2 輪車型は，2 つの車輪が回転軸を共有し，その軸を中心に動く上部の構造体（以降では本体）を軸上で支えるという構造である．重心が支点よりも上部にあるため，倒立状態を維持するには何らかの制御により安定化させる必要がある．このために必要な制御機構は，駆動のための電源を含めて，全て本体に搭載する．

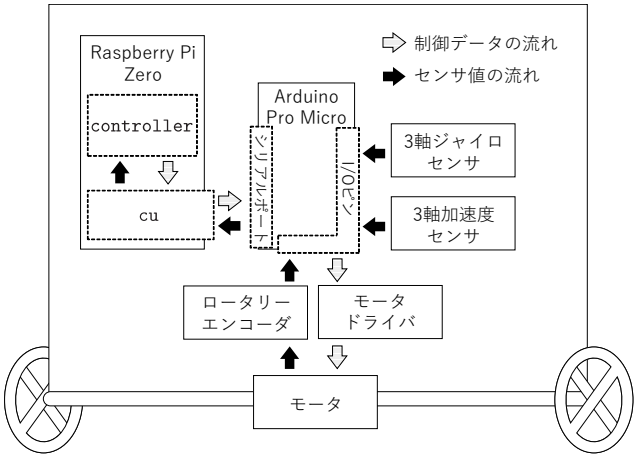

図 2. 倒立振子のシステム構成

3.3. 倒立振子のシステム構成

本実験で用いる倒立振子の構成を図 2 に示す．本システムは，UNIX 環境としての Raspberry Pi Zero（Raspbian）と，センサ・アクチュエータを統括するための Arduino Pro Micro からなる．Arduino は複数の I/O ポートを持ち，センサやアクチュエータとのやり取りを得意とする．一方で制御量の計算のような複雑な処理は Raspberry Pi で行う．Arduino と Raspberry Pi のこのような役割分担はものグラミング 2 の考え方に基づく．

ものグラミング 2 はマイコンを含むシステムに対して，適材適所の考え方を導入したものである．センサやアクチュエータは使用するためにいくつもの通信規格があるため，マイコン（Arduino）はそれら値の取得や制御に専念する．一方でセンサ値の複雑な変換処理は，豊富なリソースや柔軟な拡張が可能な UNIX 環境（Raspberry Pi）で行う．

Arduino に接続するセンサは姿勢を検出するための 3 軸加速度センサ，3 軸ジャイロセンサ，モータの回転量を測るためのロータリーエンコーダである．アクチュエータはモータを制御するためのモータドライバを使用する．Raspberry Pi とは USB で接続し，センサの値は一定間隔でシリアルに出力する．このときの出力は，各

行が特定の時刻における観測値となるような，フィールド形式 [8] をとる．つまり，ある時刻におけるセンサ値は決められた順番で，一行の空白区切りで出力される．これはフィールド形式が出力先のシェルスクリプトで処理しやすいフォーマットであるためである．シリアルから Arduino へと入力された数値データも同様にフィールド形式をとり，これはモータの速度として解釈し反映させる．

Raspberry Pi では，前述の通り，cu コマンドを通して Arduino と通信を行う．cu からの出力はセンサの値であり，図 1 に示した通り，名前付きパイプを通して制御器 C へ入力される．そして制御器で制御量を求めた後，それを cu に入力し，Arduino にモータ回転速度の指示を送る．ここまでで一連のフィードバック制御となる．

制御器には別府（2019）と同様に，線形 2 次ガウシアン（Linear Quadratic Gaussian，LQG）制御を用いる．これは制御アルゴリズムである線形 2 次レギュレータ（Linear Quadratic Regurator，LQR）にノイズ除去のための線形カルマンフィルタ（Linear Kalman Filter，LKF）を加えたフィードバック制御を指す．LKF は，センサから得られた一連のデータから，より正確な機体の状態を推定し，観測値に含まれたノイズの除去を行う．LQR は LKF で得られた機体の状態をもとに，制御量を計算する．

これら LQR，LKF の処理は UNIX 機で行う．しかし，それぞれ行うことができるコマンドが存在しないため，実装を行った．どちらのコマンドも計算に必要なパラメータが存在する．例えば，システムの状態方程式 $\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u}$ やセンサのサンプリング間隔 Δt があげられるが，必要なパラメータは全てコマンドのオプションで与える．従って，次のようなワンライナーで制御を行う[*1]．

```
$ cat named_pipe | kalfltr <options> |
 lqreg <options> | cu -l /dev/G |
 tee named_pipe > y.output
```

ここで，kalfltr が LKF コマンド，lqreg が LQR コマンドを表す．また，それらのコマンドで必要となるオプションは上記では<options>と省略しているが，実際には適切に設定する必要があることに注意されたい．これらのコマンドは流れるデータのフォーマットに合わせるため，標準入力から 1 行入力されるごとに出力値を計算し，その出力値をフィールド形式で 1 行に出力するように設計した．また，列の入替えや単位の変換のため，コマンド間に適宜 awk 等のコマンドを用いる．このとき，フルバッファリングに注意し，使用の際は ptw awk <args>のように，ptw コマンドでラップする．

[*1] 倒立に用いた実際のスクリプト：⟨https://github.com/NCNT/HerFirstSteps⟩

表 1. 倒立振子制御のためのパラメータ

パラメータ	値
Q	$\mathrm{diag}(1\ /\mathrm{rad}^2, 1\ /\mathrm{rad}^2\cdot\mathrm{s}^{-2}$ $, 1000\ /\mathrm{rad}^2, 10\ /\mathrm{rad}^2\cdot\mathrm{s}^{-2})$
R	$1000000\ /\mathrm{V}^2$
Δt	10 ms, 20 ms, 50 ms, 80 ms, 100 ms

4. 倒立振子制御の実験

本提案手法に基づいたリアルタイム制御が可能であるという実証のために，倒立振子を安定化させることを試みた．機体の設計および制御量計算のアルゴリズムは別府（2019）を基にした [17]．相違点は NUCLEO-F401RE の代わりに，Arduino Pro Micro と Raspberry Pi Zero を用いているという点である．また，それに伴った電源の変更等を行った．

実験は LKF の有無という 2 通りの制御方法によって倒立状態の安定化を試みた．比較はそれらのログ（前述したワンライナーでいう y.output）を解析して行う．

4.1. パラメータの設定

一連のシステムを動作させるために決定するパラメータは別府（2019）とほぼ同一とするが，異なる点に関しては表 1 に示す．ここで，状態ベクトル $\boldsymbol{x}$，入力ベクトル $\boldsymbol{u}$ に対して，LQR の評価関数を

$$J = \int_0^\infty \left(\boldsymbol{x}^\top Q\boldsymbol{x} + \boldsymbol{u}^\top R\boldsymbol{u}\right) dt$$

とする．また表中の $\mathrm{diag}(c_1, c_2, \ldots, c_n)$ は，(i, i) 成分が c_i であるような対角行列を表す．

実験ではサンプリング間隔 Δt を変化させ，倒立振子の安定性を評価する．

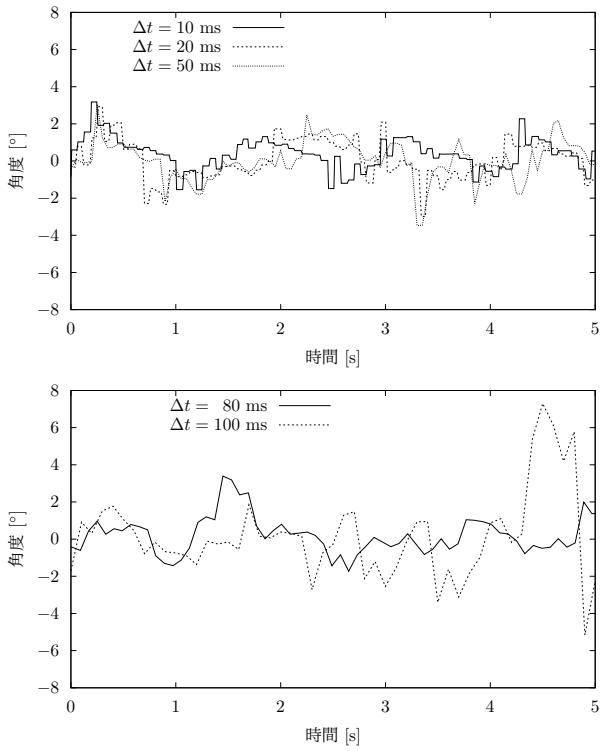

図 3. サンプリング周期を変化させた際の倒立振子の傾き推移

4.2. 倒立振子の安定性

サンプリング周期 Δt を変化させ，倒立振子の傾きの推移を測定した結果，図 3 のようになった[*2]．ここで，0° が直立状態となるように縦軸を調整し，横軸は倒立振子が自立した時刻を 0 としている．図は倒立後 5 秒間のグラフだが，Δt が 80 ms 以内であれば本倒立振子は安定し，少なくとも 1 分間倒立状態を維持することが確認できた．$\Delta t = 100$ ms のときは 4 s 以降に不安定になり倒れてしまった．

図を見ると，$\Delta t \leq 50$ ms のとき，Δt の減少に対して，倒立振子の傾きの周期が短くなっており，サンプリング間隔を短くすることが倒立振子の素早い制御に寄与していることが見て取れる．安定して倒立したとき（$\Delta t \leq 80$ ms のとき）は振幅はどれも 4° 以内であり，サンプリング周期が振幅に与える影響は見られなかった．

$\Delta t = 0.01$ ms とし，LKF を使用しない場合の，倒立

[*2] 倒立検証時の動画：〈https://www.youtube.com/watch?v=Ixh-YFtMypY〉

0.01s or 10msec

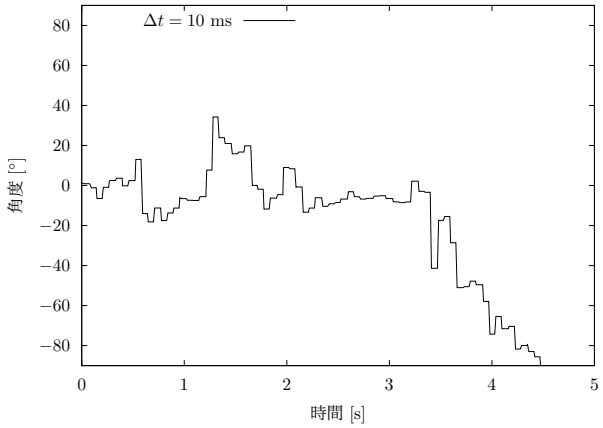

図 4. 線形カルマンフィルタがない場合の倒立振子の傾き推移

振子の挙動を図 4 に示す．LKF がない場合にはセンサのノイズを除去することができず，動作が不安定になり，5 秒持たず倒れてしまったことが見て取れる．

5. 考察

5.1. センサの値が不定期に出力される場合のリアルタイム処理

本研究で採用したシステムでは観測値の取得に Arduino を使用しており，内部のクロックを用いた等間隔の観測が可能であった．しかし，ネットワーク越しにデータを取得する場合などはこのように等間隔でデータが得られることは少ない．例えば，近年では IoT におけるリアルタイム通信のために軽量なプロトコルである MQ Telemetry Transport（MQTT）を用いることも多い [15]．しかし MQTT を使用する場合，通信による不規則な遅延や，Quality of Service（QoS）の設定によってはデータが欠損する可能性も考えられる．本システム構成のままではこのような事態に弱い．なぜならばデータが等間隔に到来するという仮定の下で，離散化したフィルタリング・制御量計算を行っているためである．

しかし，このようにデータが得られる間隔が不均一の場合であっても本手法を適用することができ，そのために `linets` コマンド [9] を使用する．これは標準入力から与えられた各行に対し，その行が入力された時間を付加し出力するコマンドである．観測値が得られた直後，

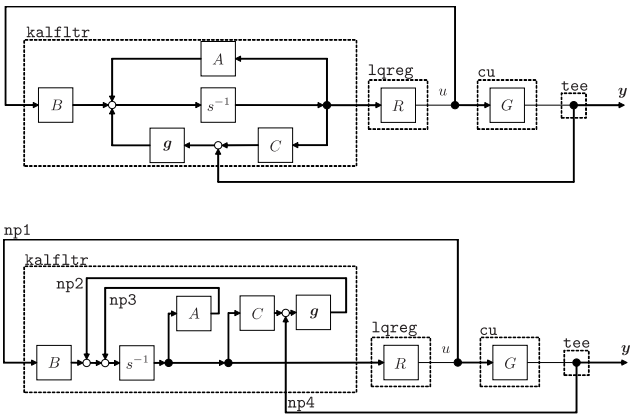

図 5. LKF を詳細に記述したブロック線図（上）とそれをシェルスクリプト適用のために書換えたもの（下）

具体的には cu コマンドの直後に linets を追加することにより，観測値をタイムスタンプ付きで後続するコマンドに渡すことが可能となる．このように linets を使用することで観測値が得られる間隔のばらつきに強いシステムを構成することが可能になると考える．

5.2. 他のブロック線図への適用

今回は図 1 の単純なフィードバック制御を実装した．しかし，このように単純化された状況は少なく，一般にはより複雑な，複数のフィードバックが混在した制御が行われる．このような場合であっても本手法が適用可能であることを示す．

図 5 は図 1 と本質的に等価だが，LKF の内部処理をより詳細に記述したブロック線図である [2]．コマンドはモジュール毎に作成されるため，さらに要素を分解し，それぞれ独立にコマンド化することは試作段階を除いて少ないが，これだけ複雑なブロック線図であっても本提案手法が適用可能であるということを示す．

まずは，次の記述方法を定める．

- 図 5 中の各ブロック内の文字を等幅フォントに直し，それをコマンド名とする．例えば A の処理をコマンド化したものを A，s^{-1} をコマンド化したものを s^-1 で表す．
- np*を名前付きパイプとする．例えば np1，np2 は名前付きパイプである．

```
$ cat np1 | B | add np2 | add np3 | s^-1 |
> A -a | tee -2 np3 | cut -d ' ' -f 1 |
> C -a | add np4 | g |
> tee -2 np2 | cut -d ' ' -f 1 |
> lqreg | tee np1 | cu | tee np3 >y.output
```

図 6. kalfltr をより単純なコマンドで置き換えた場合のワンライナー

- 各コマンドは-a オプションがない場合，入力された最終列に対して処理をし，その列を書き換える．
- 各コマンドに-a オプションを付けた場合，最終列に処理した結果をその後ろに付け加える．
- 分岐点に対応する tee コマンドに新しいオプション-n を定義する．これは n 列目を引数で与えた名前付きパイプに書き出す．
- 加え合わせ点に対応する add コマンドを導入する．これは引数で与えられたデータを標準入力から与えられたデータの最終列に加算して出力する．

このルールの下で図 5 をストリーム指向のシェルスクリプトで記述すると，図 6 のようになる．ここで，lqreg および cu のオプションは省略した．また，cut コマンドはフルバッファリングを行うため，実際には ptw コマンドの使用が必要だが，ここではそれを省略した．

スクリプトの構成法は次のとおりである．例えば，図 5（下)ではブロック s^{-1} からの出力はその直後の分岐点でブロック A とブロック R の方向に分岐している．このように並列なデータの流れはストリーム指向で記述することが難しいので，A の出力と R 方向のデータを一旦合流させる（図 6，2 行目の A -a）．その後，A の出力のみをフィードバックし（tee -2 np3），R（および C）の入力に不要な A の出力を取り除く（cut -d ' ' -f 1）．このような構成法により，並列に流れるデータをストリームという 1 次元上で扱うことができ，図 5 のような複雑なブロック線図であっても，図 6 のようにシェルスクリプトのワンライナーで表すことができる．

この構成法によって，図 1 のような単純なフィードバック制御システムに限らず，任意のブロック線図がシェルスクリプトのワンライナーで表現可能であると考えられる．

6. おわりに

本研究ではソフトウェアの移植性や持続性をこれまで以上に高めるべきという筆者らのこれまでの主張に基づき，シェルスクリプトのストリーム指向を利用したリアルタイム制御の実装手法を提案し，本手法の検証を行った．結果として，提案手法を用いて同軸 2 輪車型倒立振子の安定した倒立を LQG 制御で達成することができた．これにより，制御システムに特化した UNIX 環境でなくても，シェル上でコマンドをパイプでつなぐという UNIX 的な手法がリアルタイム制御にも有効であることが示された．

本研究では倒立振子の安定化のみ行った．しかし，倒立振子自体は外部からの指示による前後の移動に加え，2 輪を非同期に動作させれば平面上の自由な移動や坂の上り下りができることが知られている．このようなより複雑な状況下でも本手法が適用できることが示せるよう研究を継続すると共に，ドローンや 2 足歩行の制御も行えるよう本手法の検証を進める．加えて，本手法が適用可能であるためのリアルタイム性に関する条件も調査していく．

謝辞

本論文執筆にあたり，USP 研究所の矢嶋遼氏，並びに北嶋完基氏には倒立振子の開発補佐や動画の作成をしていただいた．また，矢嶋氏，北嶋氏をはじめとする同社研究部門の方々との議論は有意義なものであり，研究の示唆を得ることができた．本研究は USP 研究所と金沢大学総合メディア基盤センターの共同研究の一環として進められたものであり，USP 研究所の當仲寛哲代表には研究環境および研究資金をいただき，滞りなく研究をすすめることができた．ここに記して感謝する．

参考文献

[1] Arduino Foundation “Arduino”, https://www.arduino.cc/（2021 年 5 月 20 日閲覧）.

[2] Lachhab, A., A. Ed-Dahhak, and M. Guerbaoui “Synthesis of an optimal dynamic regulator based on linear quadratic Gaussian (LQG) for the control of the relative humidity under experimental greenhouse”, *International Journal of Electrical and Computer Engineering*, vol. 6, no. 5, pp. 2262–2273, 2016.

[3] Gancarz, M. (芳尾桂 訳) “UNIX という考え方”, **オーム社**.

[4] Garg, A. “Real-Time Linux Kernel Scheduler”, Linux Journal, 2009. https://www.linuxjournal.com/article/10165（2021 年 3 月 15 日閲覧）.

[5] Jones, M. T. “Anatomy of real-time Linux architectures”, BM developerWorks, 2008. https://www.ibm.com/developerworks/library/l-real-time-linux/index.html（2021 年 3 月 15 日閲覧）.

[6] Raspberry Pi Foundation “Teach, Learn, and Make with Raspberry Pi”. https://www.raspberrypi.org/（2021 年 5 月 20 日閲覧）.

[7] The Open Group “The Open Group Base Specifications Issue 7, 2018 edition”. https://pubs.opengroup.org/onlinepubs/9699919799/（2021 年 3 月 15 日閲覧）.

[8] USP 研究所 “Tukubai オンラインコマンドマニュアル”, https://uec.usp-lab.com/TUKUBAI_MAN/CGI/TUKUBAI_MAN.CGI?POMPA=MAN5_field-format, 2021 年 3 月 15 日閲覧.

[9] USP 研究所 NCNT プロジェクト “Timing Management Commands in POSIX”, GitHub. https://github.com/NCNT/TimingCmds_in_POSIX（2021 年 3 月 15 日閲覧）.

[10] Yodaiken, V. “The RTLinux Manifesto”, *Proceedings of the 5th Linux Expo*, 1999.

[11] 石綿陽一, 松井俊浩, 国吉康夫 “高度な実時間処理機能を持つ Linux の開発”, **日本ロボット学会学術講演会予稿集**, vol. 16, no. 1, pp. 355–356, 1998.

[12] 大野浩之, 松浦智之, 森祥寛 “ものグラミング 2 - 諸機能の選択と集中を徹底した POSIX 中心主義に基づく IoT 開発方式の提案”, **研究報告インターネットと運用技術（*IOT*）**, vol. 2019-IOT-46, no. 16, pp. 1–8, 2019.

[13] 大野浩之 他 “ものづくりのための「ものグラミング」と実践的教育環境の構築”, **マルチメディア，分散，協調とモバイルシンポジウム** *2016* **論文集**, vol. 2016, pp. 1335–1340, 2016.

[14] 熊谷正朗，森友一朗，橋本浩一 “Linux 系 OS によるリアルタイム制御”, **計測と制御**, vol. 43, no. 6, pp. 515–520, 2004.

[15] 佐々木大知, 福田浩章, 高橋和也 “製造業における製品品質のオンラインリアルタイム診断ツール”, **研究報告ソフトウェア工学（***SE***）**, vol. 2020-SE-204, no. 16, pp. 1–6, 2020.

[16] 清水正晴, 戸田健吾, 林原靖男, 大和秀彰, 古田貴之 “Linux 標準機能を利用した RT ミドルウェア周期実行機能のリアルタイム化”, **計測自動制御学会論文集**, nol. 46, no. 1, pp. 16–23, 2010.

[17] 別府伸耕 “特集　月着陸船アポロに学ぶ確率統計コンピュータ”, **トランジスタ技術** *2019* **年** *7* **月号**, 2019.

[18] 松浦智之, 大野浩之, 當仲寛哲 “ソフトウェアの高い互換性と長い持続性を目指す POSIX 中心主義プログラミング”, **デジタルプラクティス** vol. 8, no. 4, pp. 352–360, 2017.

[19] 松浦智之, 柳戸新一, 大野浩之 “UNIX 機における IoT 機器制御のためのタイミング管理”, **ソフトウェア・シンポジウム** *2021*（投稿中）, 2021.