Vol.2015-SLDM-170 No.45
Vol.2015-EMB-36 No.45
2015/3/7

RLS:分散ロボットシステムのデータ収集プラットフォーム

中山 悟[†1]　住谷 拓馬[†1]　中野 美由紀[†1]　菅谷みどり[†1]

近年, 安価なロボットが利用可能になり, 多数のロボットを利用したサービスも検討されるようになってきた. 複数台のロボットを対象とし, 作業中のデータを効率的に収集するためには, ロボットからログを収集する仕組みが必要である. 一般に従来のロボットは計算機とは異なり, ハードウェアやソフトウェアが固有であることが多く, 汎用のプラットフォームが十分に提供されていない. 本研究では, サーバを介して分散ロボットシステムから汎用的にデータを収集するためのプラットフォームを提供することを目的とした. 今回, その第一段階として, ロボットを複数台用いて, 軽量のマルチタスクサーバを設計, 実装し, 評価を行った.

RLS:The data collection platform of the distributed robotic system

SATORU NAKAYAMA[†1]　TAKUMA SUMIYA[†1]
MIYUKI NAKANO[†1]　MIDORI SUGYAYA[†1]

In recent years,inexpensive robot becomes available,since services using a large number of robots have also come to be discussed.For multiple robots,in order to efficiently collect the data in work requires a mechanism which collects logs from the robots.In general,the conventional robot is different from the computer and often has unique hardware and software,since general-purpose platform is not provided sufficiently.In this paper,to provides a platform for generic data collection from the distributed robotic system through the server as the purpose.This time,as a first step,we designed and implemented the light multi tasking server,then evaluated it with multiple robots.

1. はじめに

近年,安価なロボットが利用可能になり,多数のロボットを利用したサービスも検討されるようになってきた.特に,災害時のレスキューや燃料補給などは緊急性を必要とされるため複数台で行う事が求められる[1].また,Pepper[2]など人間とのコミュニケーションを目的としたロボットが提供されている.こうしたロボットは,フロア案内をするなどにより,人的資源が限られた店舗において,待ち行列の緩和などが期待できる.このように,複数台のロボットを用いたサービスは様々な場面での利用が期待される.

複数台のロボットを利用したサービスにおいては,サービス提供者側は,個々のロボットを管理するために,その情報収集が必要となる.ロボットは,自律制御がある,ないに関わらず,移動や会話など,動的に変化する環境への変化が求められることから,その目的の達成についてデータを収集して確認するという要求は,システム構成上自然な要求である.

例えば,エンコーダ情報による移動距離,GPS による位置情報などの情報で位置推定,衝突回避などの場合には,障害物の検知情報などは,一般的にロボットを開発する際に最初に取得される情報である.また,ロボットと人間のインタラクションを前提とする場合には,まずは,人がどのようにロボットとインタラクションをするのか,など,未だロボットが十分普及していない段階では,こうした情報の収集が重要である.このように,ロボットから情報を収集する動機は様々であるが,その要求は今後増大すると考えられる.

複数台のロボットを対象とし,作業中のデータを効率的に収集するためには,ロボットからログを収集する仕組みが必要である.一般に従来のロボットは計算機とは異なり,ハードウェアやソフトウェアが固有であることが多く,汎用のプラットフォームが十分に提供されていない.特に,多くのロボットはシリアル通信で近距離,双方向通信はサポートしている事は多いが,TCP/IP を用いた通信はサポートしておらず,サーバを介した情報収集は十分行えない問題がある.

本研究では,サーバを介して分散ロボットシステムから汎用的にデータを収集するためのプラットフォームを提供することを目的とした.今回,その第一段階として,iRobot 社の iRobot Create[8]ロボットを複数台用いて,TCP/IP を扱うための軽量のマルチタスクサーバを設計,実装し,評価を行った.ロボットのログを収集するサーバの設計にあたっては,実際にロボットを教育現場での開発情報を収集した経験から要求抽出を行うことで,具体的な課題をもとに開発を行った.

本論文の構成は,以下の通りとする. 2 節にて既存研究,3 節にてロボット API のログ収集の有効性を示し,4 節にてロボットを用いた情報収集プラットフォームの提案,5 節で評価,6 節でまとめとした.

2. 既存研究

ロギングの仕組みは,ソケット通信や,それをもとにしたサーバクライアントの仕組みを前提とした様々な手法が提案されている.例えば, UNIX マシンであれば,syslog-ng[3]

†1 芝浦工業大学　情報工学科
Shibaura Institute of Technology,Information Science and Engineering

ⓒ 2015 Information Processing Society of Japan

などが利用できる.syslog-ng は,システムメッセージをネットワーク上で転送,あるいは,ファイルに記録する仕組みである,syslog の実装の 1 つである.出力するログのフィルタリングや,ローテート,出力先ディレクトリの自動作成,ログにメタ情報などを表すマクロが使用できる.

Fluentd[4]は,インターフェイスの記述を Ruby とすることで,よりログ収集をしやすくすることを目指している.オブジェクト志向での記述が可能であるため,ログを収集する際には, Input,Buffer,Output の 3 つのコンポーネントを利用したクラスで様々なログの入力(Input) や出力(Output) に対応できるものとしている.また,収集したログを JSON 形式に変換し,蓄積した後に様々な出力先にデータ出力することで,オンメモリの処理をしやすくすることができる.このようなユーザビリティの向上から,近年多くのシステムで利用されるようになっている.

しかし,一方,こうしたツールは,ロボット上でログを収集する際には,ソフトウェアの規模が大きく,複雑である問題がある.本研究では,こうした問題に対して,より使い易くシンプルな手法を提供することを目的とする.

3. ロボット API のログ収集の有効性の評価

3.1 概要

システムの開発に先駆けて,我々は複数台のロボットから,ロボット API を利用して収集したログの有効性を測るために,まず,有効性を調査する方法を検討した.一般的に,ロボットを動作させるための開発では,基本的な動作を実現するための API を組み合わせて実行する.一般的な PC 上でのプログラミングとロボット上でのプログラミングの差異は,ロボットの個体差や物理条件に合わせてチューニングを行うことである.

そこで,我々はロボット開発者が,どのようにチューニングを行うのか,API のログを解析することにより,明らかにすることを目的とした.そのためには,ある程度の開発者が同時に開発する環境でログを取得することが望ましく,それを実現できるロボット演習の実践授業への適用を行うものとした.

3.2 実施方法

まずは,ロボットプラットフォーム上で,開発者（ここでは授業受講者）に,API を利用した開発を行ってもらうために,以下の手順でログの有効性について検討を行った.

1). 共通の課題を作成し,その課題をもとに受講者に開発を行ってもらう.

2). 受講者がそれぞれの課題に対して,どのような開発を行ったのかを確認するために,API のログをプラットフォームから収集する.

3). プラットフォームから収集した結果を解析し,開発過程の記録における API ログの有効性について議論する.具大抵には,当初の目的であるロボットの個体差や物理条件に合わせてチューニングを行う過程が得られたかどうかを検証する.今回使用するロボットとして iRobot 社の iRobot Create[5]を用いた.iRobot Create は,開発可能なライブラリである COIL（Create Open Interface Library）[6]と簡易な移動体ハードウェアを提供しており,プラットフォーム開発に適していると考えた.

3.3 課題内容

実施方法の 1)に示したように,共通の課題を設定する必要がある.短時間で成果を得るために,開発を競技形式で進めるものとした.対象が,熟練の開発者ではないことから,比較的単純な問題設定とし,短時間で効果を得ることを目的とし,チキンレースは,前進（forwardDistance()関数）のみを使って,可能な限り 2m に近い位置でロボットを停止させる課題である.forwardDistance()関数は,移動速度と移動距離を引数に指定し,移動距離分だけ前進した後,ロボットを停止させる関数である.

ロボットの動作の中でも,直進は,最も単純でありながら,地面との摩擦や,重量といった環境は個体差の影響があるため,指示した通りのパラメータで正確に動作させることが難しい.

開発者は,正確な値を設定できるように試行錯誤する必要があるが,こうした過程において,どのような API を提供することが望ましいのかを詳細に観察することができるように,拡張 API を提供し,差分を調査できるようにした.また,壁との距離を最小まで縮めるためには,発進直後に徐々に加速し,停止直前に徐々に減速するようにプログラミングすることが望ましい.これはロボットが前進する際に最大スピードで最大距離進んだ場合,停止した際の誤差が大きくなる問題を回避するために必要となる実装である（図 1）.

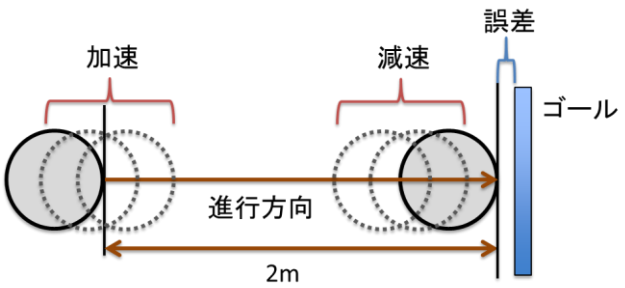


図 1　チキンレース

しかし,対象となる受講者は,必ずしもロボットプログラミングに慣れているとは限らない.そのため,ヒント無しで,こうした減速と加速を含むプログラムを書く事は難しい.そのため,競技として提示したチキンレースにおいては,図 2 のサンプルプログラムを示し,受講者がチューニングの参考にできるようにした.

ⓒ 2015 Information Processing Society of Japan

Vol.2015-SLDM-170 No.45
Vol.2015-EMB-36 No.45
2015/3/7

```
#include <createoi.h>
#include <stdio.h>
void int main() {
  int velocity = 0; //速度
  connect(); //通信開始
  // 加速処理
  while (velocity < 300)
  //速度がある値になるまでループ
  {
    if (velocity == 0) {
      velocity = 200;   //初期速度を設定
    } else {
      velocity += 50; //2回目以降，加速
    }
    forwardDistance(velocity, 100); //前進
  }
  // 前進
  forwardDistance(velocity, 1400); //前進
  // 減速処理
  while (velocity > 0) //速度が0(止まる)まで
ループ
  {
    velocity -= 50;
    forwardDistance(velocity, 100);
  }
  disconnect(); //通信終了
}
```

徐々に加速

徐々に減速

図 2　サンプルプログラム

受講者に提示したサンプルプログラムでは，まず，少しずつ加速を行うために，while ループ内で初期速度 200 から最高速度 300 になるまで速度を 50 ずつ増やしながら，距離 100 ずつ前進する．ここで，初期速度と最高速度は，実測し，誤差が小さく，かつ，滑らかに走行する速度を選んだ．次に，停止前に少しずつ減速を行うために，while ループ内で最高速度 300 から速度 0 になり停止するまで速度を 50 ずつ減らしながら，距離 100 ずつ前進するものとした．

3.4 実施概要

課題にあるプログラムを参考にして，実際に複数人にロボット上での開発を並行的に実施するものとした．実施にあたっては，開発ログを収集し，これらのログを比較検討することを目的とした．

複数人による同時開発を実現するために，ロボットを用いたプログラミングの授業で，2.3 節に示した課題を実施し，ログを収集するものとした．授業は，芝浦工業大学，工学部，情報工学科，プログラミング演習Ⅰの実習において 1 年生 108 人で，約 10 人ずつ 10 グループに分けて実施した．また，比較対象ために，同情報工学科，3 年生の別の選択必修の授業にて約 80 人を 10 人ずつ 8 グループに分けてそれぞれ実施した．

はじめに，ロボットを動作させるために必要な API を一通り学習させ，各グループにロボットと開発環境をもつパソコンを用意し，2.3 節の課題を作成させた．プログラムをコンパイルするごとにログを出力し，ファイルに追記処理した(図 3)．

1回の試行	1	Jun 30 20	10:45:17	1806	connect		
	1	Jun 30 20	10:45:17	1909	forwardDis	300	1000
	1	Jun 30 20	10:45:17	1822	disconnect		
	1	Jun 30 20	10:45:17	1806	connect		
	1	Jun 30 20	10:45:17	1909	forwardDis	300	500
	1	Jun 30 20	10:45:17	1822	disconnect		

グループ番号　日付　時刻　関数の行番号　実行した関数　関数の引数

図 3　出力されたログ

3.5 分析結果

各グループから得られたデータの中から，直進の加速，減速を想定したチキンレースで使われた forwardDistance() 関数の速度パラメータを抽出し，コンパイル時間ごとの推移を図 4, 5 に示した．

図 4 は，ある固定値で単純な試行を行ったグループの内から，また，図 5 は，サンプルプログラムに習って，加速と減速を用いて試行を行ったグループの内から，最も典型的なものを選んだ．また，図 4 において，ロボットの速度は 0 から 500 の値の範囲であるが，グループによっては，速度 1000 で試行を行っているものもあった．

ログからは，forwardDistance() 関数を 1 つだけ使ってプログラム作成したグループと，forwardDistance() 関数を複数使って加速と減速をするプログラムを作成したグループの 2 パターンに分類する情報を得ることができた．

そのうち，加速と減速をするプログラムを作成したグループは，1 年生では 10%，3 年生では 37.5%となった．1 年生は短時間でソースコードを理解して読み進める力が弱く，サンプルコードを解釈して適切に拡張することが難しかったことが読み取れる．それに対して，3 年生は開発力がつき，ソースコードを短時間で理解して，拡張することができたチームが増えたことが分かる．しかし一方で，こうした事ができたチームは全体の割合からいうと決して多くはない事から，今後の開発力改善のためにはさらなる実践演習等の機会が必要であると考えられる．

このように，ログを解析することにより，開発過程に差があることが分かった．今回は，情報工学科 1 年生と 3 年生に同じ課題を出したことから，その開発にあたり，示されたサンプルを解釈実行する開発能力に差があったといえる．このログの分析により，学部の教育の有効性について議論することも可能であるといえる．

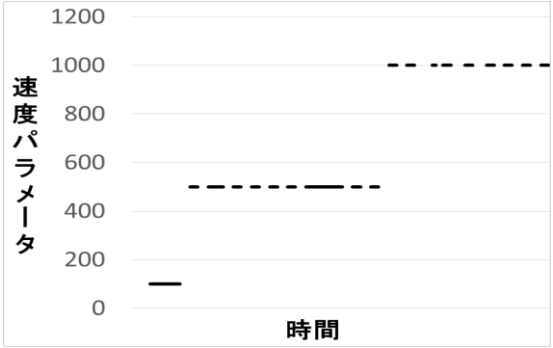


図 4　加速・減速を考慮しないプログラムの開発過程

ⓒ 2015 Information Processing Society of Japan

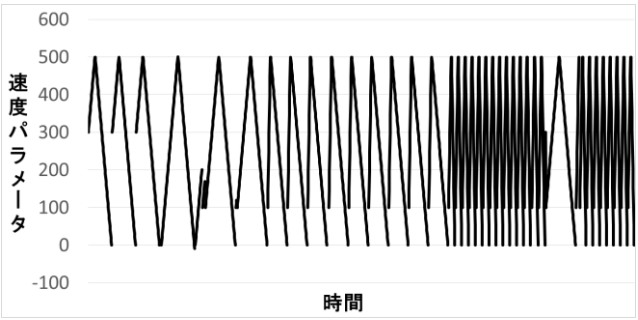


図 5　加速・減速を考慮したプログラムの開発過程

3.6 考察

API のログを解析することで,プログラムの開発過程や,また比較により開発力の全体の傾向を知ることができることが分かった.このことから,ログを取得し,開発過程を理解することで,異なる物理条件下におけるロボットのチューニング過程を記録し,共有することが重要であると考えられる.

3.7 課題

本実験では,実行するロボットプログラムを作成した計算機上にログファイルを出力したため,全てのグループがチキンレースのプログラムを作成し終えた後に手作業でそれぞれのログファイルを収集する必要があった.この手間を省くとともにリアルタイムでログを収集するためにソケット通信プログラムを基にしたモニタとサーバプログラムの作成を検討した.

4. 提案

4.1 システム構成

ロボットは,制動部の制御の計算を計算機と接続して行うことが一般的である.そこで,今回はまず,ロボットを計算機と接続し,計算機側で情報を収集し,処理する役割を担わせ,そこから命令を発してロボットを動作させる構成とした.また,この際に,ロボットは主に計算機とシリアル接続していることを前提とした.データの送受信は,ロボットのログを集約するサーバを想定し,そのデータをサーバに集約するものとした(図 6).

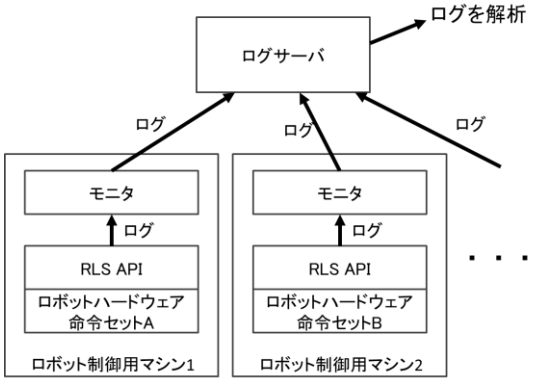


図 6　システム構成

ロボットとして iRobot 社の iRobot Create を用いた.システム構成においては,計算機とシリアル通信を行うものとした.

通信では,計算機側から,実行命令,また,ロボット側からは実測値を得るものとした.ロボットは,計算機から送られた命令実行後に,その実行時間（日時）とともに,実行命令内の理想値と,実測値を計算機内のログファイルに保存する.ここで,理想値とは,実行命令である API の引数であり,実測値とは,ロボットが実際に命令実行した後に返されるセンサ値である.次に,ログの更新を監視するモニタアプリケーションは,更新されたログファイルの日時と内容を監視し,更新され次第,書き込まれたデータを TCP/IP 通信でサーバに送信する.複数台からのデータを受け取ったサーバは,それぞれでサーバ内のファイルにデータを保存する.

4.2 API 設計と実装

ロボットの動作を観察するためのログの収集にあたり,iRobot Create のオペコードを C 言語で隠蔽するライブラリである COIL（Create Open Interface Library）を使用することで,POSIX 準拠の OS において C 言語でプログラムを記述しロボットを動作させるものとした.

ログ出力の際には,API のみで物理的な動作ができるだけ明確にわかることが重要である.COIL で提供されている API は,汎用性が高い一方,初心者が動作させる場合には,物理的な動作がイメージしづらいものであった.そのため,本研究では,ログに取得することも考慮して,API の拡張を行った.拡張した結果は,表 1 にまとめた.

表 1　設計した API とその役割

関数名	動作
begin/end()	iRobot Create との通信を確立,または,切断する.
forwardDrive/ backwardDrive (short vel, int dist)	一定の速度(vel)で指定した距離(dist)前進,または,後退した後に停止する.dist を 0 に設定することで waitTime(double time)による時間制御が可能となる.
forwardTurn/ backwardTurn (short vel, short rad, int angle)	一定の速度(vel)と回転半径(rad)で指定した角度(angle)回転した後に停止する.angle を 0 に設定することで waitTime(double time)による時間制御が可能となる.

ここで,vel は 0 から 500 の値の範囲を取る.これは,もとの COIL の API の値の範囲では,-500 から 500 であったが,負数を取らないようにする代わりに,前進と後退それぞれ別々の API を用意することで正数のみを取るものとした.同様に, また,dist , angle, rad は,もとの COIL の API の値の範囲を改変せずに利用するものとした.

実装した API と,元にした COIL の API との相違点は表 2 の通りである.

ⓒ 2015 Information Processing Society of Japan

Vol.2015-SLDM-170 No.45
Vol.2015-EMB-36 No.45
2015/3/7

表 2　元の COILAPI との比較

API	元の COILAPI	変更点
begin/end()	startOI_MT/ stopOI_MT (const char* serial)	引数をそれぞれの計算機のシリアルデバイスファイル名で指定する形式であったが,計算機ごとに固定値を設定し,ユーザーはファイル名を気にせずアクセスできるようにした.
forwardDrive/ backwardDrive (short vel,int dist)	driveDistance (short vel, short rad,int dist,int interrupt)	直進のみを扱うようにした. 前進と後退で別々の API にした. ロボットを持ち上げた際に API の実行を停止するよう固定した.
forwardTurn/ backwardTurn (short vel,short rad,int angle)	turn (short vel,short rad,int angle, int interrupt)	回転する半径と角度により,進行方向を変えるようにした. ロボットを持ち上げた際に API の実行を停止するよう固定した.

このように,COIL 本来の API 設計を修正することで,開発時の内容の詳細が把握しやすく,また,本 API を利用する側もより書き易くすることを目指した.

4.3　API のログ収集の仕組み

API のログを収集するにあたり,さらにロボットが実行した API を収集するために,下記のログの API を設計,実装した.各ログ API を図 7 にまとめた.

functionLogger(const char* funcName, const int argNum, …)

sensorLogger(const int argNum, …)

functionLogger()は,funcName で呼び出された関数名,argNum で引数の数,その後に引数として指定された実際の値を出力する.API 名の取得にはマクロ__func__を使用することで開発者は,呼び出す API を記載せずに利用できるものとした.また, 動作指示の命令を与える API を実行した結果, ロボットが動作した際には,その実測値を得るために sensorLogger()を設計した.sensorLogger()は,物理的な動作を行う命令の実行前と, 実行後に呼び出すことで,その物理的な変化を実測値として出力する.iRobot Create には様々なセンサが搭載されており, 現状は距離, 角度, 速度, 回転の半径の値を取得できることから,こうしたセンサ値の取得は重要であると考えた.

また, それぞれのログ API 内で,タイムスタンプを使い,動作指示の命令が実行された日時を取得するものとした.

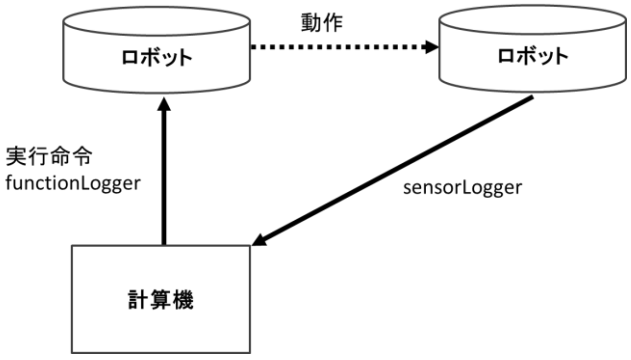


図 7　ログ API

4.4　マルチタスクサーバの設計

マルチタスクサーバの設計,実装に先立ち,ソケット通信プログラムを作成し,複数台からの処理要求を想定した簡易的な実験を行った.実験では,複数のデータ送付元が同時に,サーバにデータを送付する実験を行った.

その結果,サーバは,先に到着した処理要求に対する処理が終了するまで,その後に到着したクライアントからの要求に応じないことが確認された.このブロック時間は,ログの大きさが大きくなるほど長くなるという一般的な現象を確認することができたため,サーバ側の処理を,その都度プロセスを生成するマルチタスクにすることでそれぞれの要求の同時処理を可能とするものとした.

5.　評価

5.1　実施概要

本システムを複数台の計算機に実装し,ロボット制御用マシンとログサーバ間の通信性能を評価することを目的とし,複数台のロボット制御用マシンからログサーバに接続し,それぞれのロボット制御用マシンからログデータを送信する実験を行った.実験環境を下記に記す.

表 3　ログサーバ性能

CPU	Intel Celeron 2.0GHz
メモリ	2GB
OS	Ubuntu 14.04

表 4　ロボット制御用マシン性能

CPU	Intel Core i3 2.1GHz
メモリ	4GB
OS	Ubuntu 12.04

5.2　実施方法

本システムを実装した計算機を 4 台用意し,1 台をログサーバ,残りの 3 台をロボット制御用マシンとした.3 台のロボット制御用マシンから同時にログサーバに文字列を送信し,ログサーバから送り返された文字列をロボット制御用マシンが受け取るまでの応答時間を 20 回計測した(図 8).

ⓒ 2015 Information Processing Society of Japan

ロボット制御用マシンがログサーバへログを送信した時刻を t1 とし, ログサーバからロボット制御用マシンが文字列を受け取った時刻をそれぞれ tm1, tm2, tm3 とする. 各ロボット制御用マシンの応答時間(tm1-t1), (tm2-t1), (tm3-t1)を計測した. 本実験以外は使わないローカルネットワークを使用し, ネットワークの負荷は一定であるとする. 各試行において, ログサーバが初めに受け付けたロボット制御用マシンをマシン 1, 最後に受け付けたロボット制御用マシンをマシン 3 とした. 各ロボット制御用マシンの応答時間の平均, 最悪値, 標準偏差を図 9 に示した.

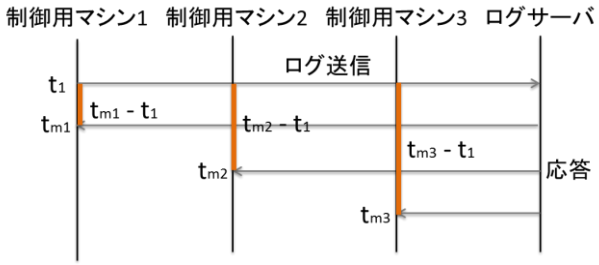


図 8　実験環境

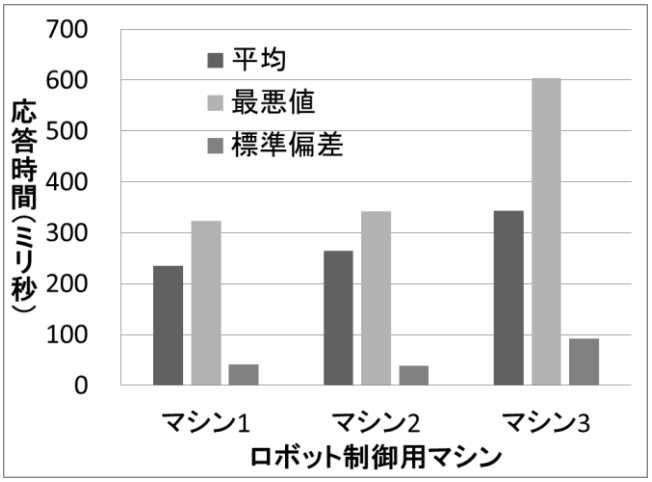


図 9　実験結果

ログサーバがロボット制御用マシンを受け付けた順序により, 応答時間に差が見られた. また, マシン 1 とマシン 2 の応答時間はほぼ等しかったが, マシン 3 の応答時間が他の制御用マシンに比べて約 1.5 倍になっている場合が見受けられた.

5.3 考察

3 台のロボット制御用マシンから送信された, それぞれの文字列はログサーバ側のログファイルへの保存は問題なく行われた. このことからログサーバがログを受け取って, ファイルに記録する部分は問題なく動作していたことが分かる. ログサーバの処理時間は, データ増大とともに増加することも考えられるため, 今後, よりデータを増やした計測が必要である.

また, ログサーバ側の処理ではロボット制御用マシンを受け付け, 親プロセスでログファイルをオープンした後, 子プロセスを生成し, 送られてきた文字列をファイルに書き込む設計とした. 設計では, ファイルをクローズするタイミングは子プロセスに任せている. この時間の差分が生じた理由として, ログサーバ側のマシン構成が理由であると考えられる. 今回, ログサーバマシンは, 2 コアのマシンを用いた. そのため, ログサーバは 2 つのロボット制御用マシンからの接続要求を, 並行して 2 つのコアで行ったと考えられる. そのため, マシン 1, 2 はほぼ同時に終了している（図 9）. その後, マシン 3 は, どちらかのコアが空いた後に実行されているため, 応答時間が遅くなったと考えられる.

6. まとめ

ロボットを対象としたデータ収集のためのプラットフォームを開発した.しかし,複数台のロボット制御用マシンを用いてログサーバにログを送信する場合に,各ロボット制御用マシンの応答時間に偏りが見られた.今後は非同期処理を実装し,ログ収集のリアルタイム性の向上を行う.例として,自然な会話に期待される応答性は 1 秒程度である[7][8].これに対して,ログ収集においてはリアルタイム性を阻害しないことが求められる.よって,システムに必要なリアルタイム性を調査し,それを阻害しないような周期でのログ収集が必要である.

参考文献

1) レスキューロボットコンテスト.
http://www.rescue-robot-contest.org/
2) ソフトバンク: Pepper ロボット.
http://www.softbank.jp/robot/
3) N.Ando, T.Suehiro, K.Kitagaki, T.Kotoku, and W.-K.Yoon,RT-Middleware: Distributed Component Middleware for RT (Robot Technology)”,Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems,pp.3555-3560,2005.
3) Official Syslog-ng site.
http://www.balabit.com/network-security/syslog-ng/
4) Fluentd | Open Source Data Collector.
http://www.fluentd.org/
5) iRobot Corporation.iRobot Create OPEN INTERFACE.
http://www.irobot.com/filelibrary/pdfs/hrd/create/Create%20Open%20Interface_v2.pdf
6) libcreateoi - COIL: Create Open Interface Library - Google Project Hosting.
https://code.google.com/p/libcreateoi/
7) Toshiyuki Shiwa,Takayuki Kanda.
"How Quickly Should Communication Robots Respond?".
HRI'08 Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction, 153-160, (2008)
8) 野呂影勇, 人間工学における反応時間の測定と結果の解析, 人間工学, 21 (2), 65-70, 1985.

ⓒ 2015 Information Processing Society of Japan