Vol.2012-GN-82 No.28
Vol.2012-CDS-3 No.28
2012/1/20

マクロプログラミングを用いた複数ユビキタス機器の集中型・分散型制御システム

長岡佑典[†1]　佐野渉二[†2]
寺田努[†1,†3]　塚本昌彦[†1]

近年，発光ダイオード (LED: Light Emitting Diode) からなるイルミネーションやディスプレイなどを用いた電飾アートが注目されている．大規模な電飾を光らせるためには大量の LED を多くのマイコンを用いて制御する必要がある．筆者らは，これまでに複数のユビキタス機器を統合的に扱うマクロプログラミングモデルを提案し，これを電飾アートに適用してきた．マクロプログラミングシステムでは，ユビキタスデバイス全体を制御するために記述された 1 つのプログラムから個々のユビキタスデバイスの制御プログラムを生成するが，システム要件に合わせて，個々のデバイスの制御方式を変えることが求められる．そこで，本稿では，多数のユビキタスデバイスの制御方式として集中型制御方式，分散型制御方式，中間型制御方式について考え，設計，実装を行った．実際にいくつかの作品を作成することで，システム要件に応じて適する制御方式が異なることを確認した．

Centralized and Distributed Control System using a Macroprogramming Model for Multiple Ubiquitous Devices

Yusuke NAGAOKA,[†1] Shoji SANO,[†2]
Tsutomu TERADA[†1,†3]
and Masahiko TSUKAMOTO[†1]

Recently, illuminations and information displays using LEDs have attracted a great deal of attention. It is necessary to control a large number of LEDs using many microcomputers to realize large-scale illuminations. We proposed a macroprogramming model for designing illuminations to control blinking patterns flexibly. But it is necessary to change control systems by system requirements. In this paper, we propose centralized and distributed control system using a macroprogramming model.

1. はじめに

近年，さまざまなモノにコンピュータを内蔵して利用するユビキタスコンピューティング[1] が注目されている．ユビキタスコンピューティングでは，複数のセンサや LED などの入出力機器を制御するデバイス (以下，ユビキタスデバイス) を連携させることで高度な機能を実現する．多数のユビキタスデバイスを統合的に制御する 1 つの手法としてはマクロプログラミング[2)–6)] がある．マクロプログラミングシステムでは，ユビキタスデバイス全体に対するプログラムをマクロな視点で記述でき，個々のユビキタスデバイスの動作を考慮しなくてもよいため，使用者はユビキタスデバイス全体に対する動作記述に注力できる．

一方で，近年発光ダイオード (LED: Light Emitting Diode) からなるイルミネーションやディスプレイなどを用いた電飾アートが注目されている．しかし，大規模な電飾アートをマイコンなどを用いて柔軟に制御するためには，多くのマイコンを統合的に制御する必要があり，特に 1 つ 1 つのマイコンにプログラムを書く必要がある場合，電飾の発光パターンの変更には多大な労力と時間がかかる．

筆者らは，これまでに複数のユビキタス機器を統合的に扱うマクロプログラミングモデルを提案し，これを電飾アートに適用してきた．提案モデルでは，ユビキタスデバイスの位置関係に基づいて，センサや LED などの入出力機器の制御プログラムを記述するだけで，環境内に存在する複数のユビキタスデバイスの個々プログラムが生成され，配置・実行される．このとき，通信量や各ユビキタスデバイスで行う処理の負荷などのシステム要件に応じて個々のユビキタスデバイスの適した制御方式を決定して，個々のプログラムを生成する必要がある．そこで，本稿では，ユビキタスデバイスの制御方式として，それぞれのユビキタスデバイスで行う負荷のバランスに着目し，集中型制御方式，分散型制御方式，中間型制御方式を考える．集中型制御方式は 1 つのユビキタスデバイスにそれ以外のユビキタスデバイスの制御を管理させる方式であり，運用管理がしやすい反面，通信量が多い，全体を管理する 1 つのユビキタスデバイスへの処理の負荷が大きい．分散型制御方式では，各ユビキタスデ

†1 神戸大学大学院工学研究科
Graduate School of Engineering, Kobe University
†2 神戸大学大学院自然科学研究科
Graduate School of Science and Technology, Kobe University
†3 科学技術振興機構さきがけ
PRESTO, Japan Science and Technology Agency

Vol.2012-GN-82 No.28
Vol.2012-CDS-3 No.28
2012/1/20

バイスが独立して入出力処理を管理する方式であり，各ユビキタスデバイスの処理の負荷が均等である一方，一部のユビキタスデバイスが故障時にその故障個所を特定するのが困難である．中間型制御方式では，全体を制御するユビキタスデバイスを換えたり，ほとんどの入出力制御は分散型制御方式を適用するが，一部の制御においては集中型制御方式を適用するなど，ユビキタスデバイスの負荷については，集中型制御方式，分散型制御方式の中間に当たる制御方式である．集中型制御方式，分散型制御方式の長所を取り入れた制御を行える．

本稿は以下のように構成されている．第 2 章で関連研究について述べ，第 3 章で複数ユビキタス機器を統合的に扱うためのマクロプログラミングモデル，第 4 章で集中型制御方式，分散型制御方式，中間型制御方式について説明する．第 5 章で考察を行い，最後に第 6 章でまとめを行う．

2. 関 連 研 究

多くのコンピュータの制御を 1 つのプログラムで行うマクロプログラミングの研究はこれまでにも多数行われている．Gumadi らの Kairos[3] は，多くのコンピュータに対してマクロな視点で個々のコンピュータを制御することに着目し，複数コンピュータにまたがる制御やコンピュータ間のネットワークトポロジを用いた制御を単一のプログラム上で記述できる．Newton らの Regiment[4] では，各コンピュータが取得するセンサなどのデータを関数型プログラミングの記述で扱える．Bischoff などの RuleCaster[5] は，簡単なルール形式の記述により，多くのコンピュータにまたがる処理を行うことができる．これらの研究では，システムはコンピュータ群に対して記述したプログラムを個々のコンピュータ用のプログラムに変換，分配する機能を有し，使用者はコンピュータ全体に対する記述をするだけでよい．

大規模な電飾アートを容易に制御可能なシステムとして，木下らの電飾アートの制御支援システム[7] がある．このシステムでは，大規模な電飾アートを小単位に分割して分散制御し光り方の変更や故障に対する柔軟性を高めている．また，簡単なコマンド入力により，直観的に LED の点滅パターンの設計を行える．一方で，出力のみの制御であるため，センサなどを用いた入力による LED の点滅パターンの設計ができないといった問題がある．大量の LED を用いた電飾の制御として，中田らのプロジェクタとユビキタス光デバイスを用いた電飾制御[8] がある．これは，プロジェクタから照射された光を，ユビキタスデバイス上の光センサが感知し，LED を点灯させている．点滅パターンの変更は，プロジェクタで照射する光のパターンを PC で変更するだけで行えるため，プログラムの書き換えやユビキタスデバイス間の通信が必要ない．また，プロジェクタの照射できる範囲であればユビキタス光デバイスの制御が行えるため，数百個，数千個の LED の一斉制御ができる．しかし，光センサには指向性があるため，プロジェクタの照射角度や照射光の強弱の影響を受けやすく，設置条件が厳しいという問題がある．さらに，球体などプロジェクタ光の影になる部分を光らせたい場合やプロジェクタの照射範囲を越えるような大規模なもので用いることも難しい．

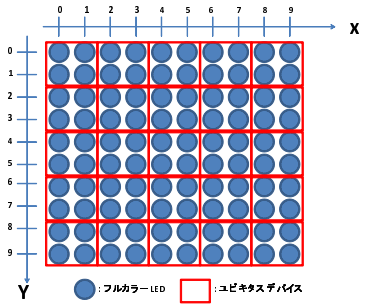

図 1 RGB フルカラー LED と一体となったユビキタスデバイスを格子状に並べた電飾アート

3. 複数ユビキタス機器を統合的に扱うためのマクロプログラミングモデル

本研究では，電飾アートとして図 1 のように複数のユビキタスデバイスを利用して格子状に並べられた多くのフルカラー LED を想定する．筆者らはこれまでに 1 つのユビキタスデバイスで複数の LED を制御するマクロプログラミングモデルを提案している[9]．マクロプログラミングを用いることで，複数のユビキタスデバイスに対して，それらの制御方法についてのみ着目して記述するだけでよく，複数のユビキタスデバイス全体について記述された 1 つのプログラムを用いて複数のユビキタスデバイスを制御する．

3.1 電飾アートのためのプログラミング言語設計

電飾アートを作成するためのプログラミング言語として Processing[10] を用いる．本研究では，LED の位置に基づいて線や三角，四角，丸のような図形を描く関数を用いて LED 群を制御する．これらの関数には Processing であらかじめ用意されている関数をマクロプログラミング向けに拡張したものや光の制御を直観的に行うために新たに作成した関数が含まれる．以下，拡張した関数と新たに作成した関数について説明する．関数内のパラメータについては表 1 にまとめた．

3.1.1 位置に基づく光の制御を行うための関数

格子状の電飾アートの光の制御を行うための関数について説明する．

Vol.2012-GN-82 No.28
Vol.2012-CDS-3 No.28
2012/1/20

表 1 関数内のパラメータ

ID	ユビキタスデバイスの ID 番号
mode	IO ピンの INPUT/OUTPUT の設定
pin	ピン番号
value	HIGH または LOW，アナログピンの場合は 0〜255 の値
x	x 座標
y	y 座標
width	幅
height	高さ
r	半径
color	LED の色 (0:白 1:赤 2:緑 3:青 4:桃 5:黄 6:水色)

- **colorMode**(*color*): 点灯する LED の色を color で設定する．他の描画関数において色の指定が行わなかった場合にはここで指定した色が用いられる．
- **point**(*x, y, color*): 点 (x, y) に位置するユビキタスデバイスの LED を color の色で点灯させる．
- **line**(*x1, y1, x2, y2, color*): 点 (x1, y1) と点 (x2, y2) を結ぶ線上に位置するユビキタスデバイスの LED を color の色で点灯させる．
- **triangle**(*x1, y1, x2, y2, x3, y3, color*): (x1, y1)，(x2, y2)，(x3, y3) の 3 点を結ぶ三角形上に位置するユビキタスデバイスの LED を color の色で点灯させる．
- **rect**(*x, y, width, height, color*): 点 (x, y) を左上の角として幅 width，高さ height の長方形上に位置するユビキタスデバイスの LED を color の色で点灯させる．
- **fill**(*color*): 図形の塗りつぶし部分上にある LED を color の色で点灯させる．
- **noFill**(): 図形の塗りつぶし部分上にある LED を消灯させる．
- **stroke**(*color*): 線や図形の枠線上にある LED を color の色で点灯させる．
- **noStroke**(): 線や図形の枠線上にある LED を消灯させる．

以下は新たに作成した関数について説明する．

- **circle**(*x, y, r, color*): 点 (x, y) を中心とした半径 r の円上に位置するユビキタスデバイスの LED を color の色で点灯させる．
- **Low**(*x, y*): 点 (x, y) に位置するユビキタスデバイスの LED を消灯させる．
- **allHIGH**(*color*): すべての LED を color の色で点灯させる．
- **allLOW**(*color* **or ALL**): 指定の色のすべての LED を消す．ALL の場合は色に関係なく全ての LED を消す．
- **Read**(*x, y*): 点 (x, y) に位置する LED に接続されたユビキタスデバイスのセンサの値を読み取る．

4. マクロプログラミングのための集中型・分散型制御方式

前章では，1 つのプログラムでユビキタスデバイス群を制御するマクロプログラミングモデルについて述べた．マクロプログラミング環境では，ユビキタスデバイス群全体に対して記述されたプログラムから個々のユビキタスデバイスで動作するプログラムを生成して実行される．このユビキタスデバイス群全体に対するプログラムをグローバルプログラムと呼び，個々のユビキタスデバイスで動作するプログラムをローカルプログラムと呼ぶ．ユビキタスコンピューティングで求められる以下の要件を満たしながらローカルプログラムを生成する必要がある．

- 同期制御
  複数のユビキタスデバイスで 1 つの電飾アートを制御するため，異なるユビキタスデバイス上の複数の LED を同時に光らせたい場合，LED 点灯のタイミングを合わせる必要がある．特に，頻繁に点滅を繰り返すような表現の場合，少しでも点灯のタイミングがずれると電飾アートの表現力が低下する可能性がある．そのため，動作のタイミングを常にそろえる同期制御が求められる．
- ユビキタスデバイスで行う処理の負荷
  ユビキタスデバイスは小型であるため，動作周波数やメモリ量が制限される．そのため，それぞれのユビキタスデバイスに処理の負荷がかかりすぎないように行うことが求められる．
- 運用管理
  多くのユビキタスデバイスを使用するほど，ユビキタスデバイスの物理的な故障や通信の不具合が起こる可能性は高まる．長時間の運用を考えると一部のユビキタスデバイスが故障しても修復するためにその故障箇所をできるだけ早く把握することが求められる．
- 通信量
  ユビキタスデバイス間で協調して動作を行うためには，メッセージを通信する必要がある．しかし，特に無線通信を行う場合，通信量が多くなるとデータの衝突などによるパケットロスが生じる可能性が高まり，正常な協調動作が行われない可能性がある．また，一般にユビキタスデバイスは電池駆動であるため，通信量が多くなると，通信に要する消費電力が大きくなり，長時間駆動に適さない．このため，通信量を削減すること

Vol.2012-GN-82 No.28
Vol.2012-CDS-3 No.28
2012/1/20

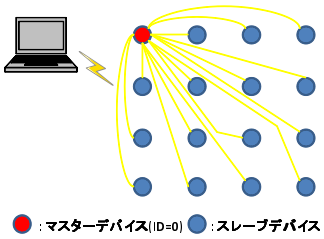

図 2 集中型制御方式

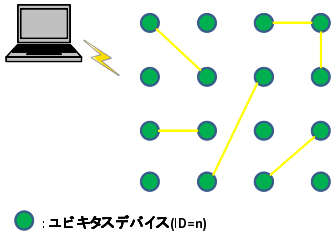

図 3 分散型制御方式

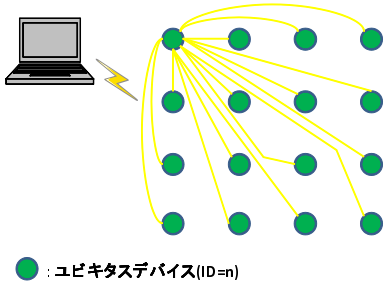

図 4 中間型制御方式

が求められる．

本稿ではこれらの要件を考慮し，ユビキタスデバイス群の制御方式として集中型・分散型・中間型制御方式を提案する．

4.1 集中型制御方式

1 つのデバイスでグローバルプログラムの処理の流れを管理し，命令を送ることで他のユビキタスデバイスの入出力を制御する方式を集中型制御方式（図 2）と呼ぶ．グローバルプログラムの処理の流れを管理するユビキタスデバイスをマスタデバイス，それ以外のユビキタスデバイスをスレーブデバイスとする．マスタデバイスはグローバルプログラムに沿って処理を行い，スレーブデバイスの入出力を制御する場合はそのデバイスに命令を送る．スレーブデバイスはマスタデバイスから命令を受けるとその命令に応じて入出力を制御する．1 つのユビキタスデバイスがユビキタスデバイス群全体の出力や入力値を扱うため，同期をとりやすい．また，すべての入出力がマスタデバイスとスレーブデバイスでのメッセージのやり取りで行われているため，正常に動作しないスレーブデバイスがあるとそのスレーブデバイスが不具合を生じさせているとすぐ把握でき，運用管理がしやすい．しかし，スレーブデバイスの入出力のたびにマスタデバイスがそのスレーブデバイスにメッセージを送って命令する必要があるため通信量が多くなる．また，マスタデバイスの処理の負荷が大きくなってしまい，マスタデバイスが故障したときユビキタスデバイス群全体が動作しなくなる．

4.2 分散型制御方式

グローバルプログラムに対して各ユビキタスデバイスで行う処理のみに振り分け，各ユビキタスデバイスが独立して入出力を制御する方式を分散型制御方式（図 3）と呼ぶ．他のユビキタスデバイスのセンサデータなどの入力値が必要な場合はそのデバイスからメッセージを送ってもらうことで取得する．分散型制御方式では，集中型制御方式のマスタデバイスのように全体の処理を管理するデバイスはなく，各ユビキタスデバイスがそれぞれ必要に応じて同期をとりながら処理を行うことで，ユビキタスデバイス全体の動作としてグローバルプログラムで記述された通りに処理が行われているように見える．このため，集中型制御方式のマスタデバイスとスレーブデバイスのように常に通信を行う必要がなく，通信量が少なくて済む．また，各々のユビキタスデバイスが独立して制御をするため，各ユビキタスデバイスの処理の負荷が分散され，著しく偏る場合は少なくなる．しかし，ネットワークトポロジはグローバルプログラムに応じて変わるので，正常に動作しない場合，どの箇所で不具合が生じているか特定しにくい．また，すべての処理で同期をとられない場合もあり，グローバルプログラムで記述された動作とは局所的にタイミングがずれる場合がある．

4.3 中間型制御方式

ユビキタスデバイス間の処理負荷のバランスが，集中型制御方式と分散型制御方式の間にある制御方式を中間型制御方式（図 4）と呼ぶ．集中型制御方式では，マスタデバイスの処理の負荷がスレーブデバイスより大きく，分散型制御方式では，各ユビキタスデバイスで行う動作に応じて処理の負荷が決まるが，上述のようにシステム要件に対して一長一短である．中間型制御方式では，システム環境に合わせて，集中型制御方式と分散型制御方式の特徴を併せもった制御を行える．例えば，運用管理を容易にしたいが，制御するユビキタスデバイスの数が増え，1 つのマスタデバイスで処理の負荷が大きくなりすぎる場合は，部分的にマスタデバイスを入れ替えて処理を行うようにしたり，同期制御を重視しつつも通信量をできるだけ抑えたい場合は，入出力制御はそれぞれが独立して行い，同期をとるためのメッセージ通信は 1 つのデバイスから発することで同期をとりやすくするなどが考えられる．

4.4 実 行 環 境

本研究では，ユビキタスデバイスとして図 5，6 に示すようなフルカラー LED4 個と加

Vol.2012-GN-82 No.28
Vol.2012-CDS-3 No.28
2012/1/20

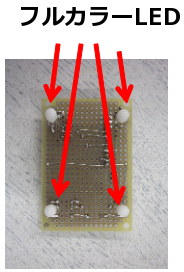

図 5　フルカラー LED 面

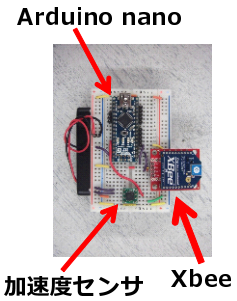

図 6　Arduino nano 面

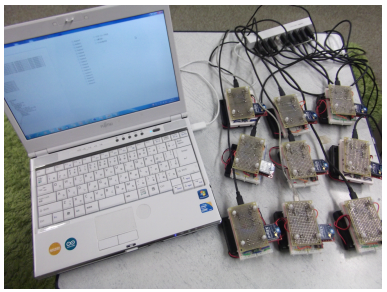

図 7　電飾アートの制御風景

速度センサ，無線通信モジュールの Xbee を Arduino nano[11] に搭載したものを用いる．Arduino nano とは，マイクロコントローラ (AVR) と入出力ピンなどを搭載した基板である．加速度センサは電飾アートへのインタラクションのために用いる．各ユビキタスデバイス上の Arduino nano の ID はあらかじめ割り当てられているものとし，ユビキタスデバイスへのプログラムのアップロードは図 7 のように 1 つの PC で行う．

マクロプログラミングでは，グローバルプログラムで個々のユビキタスデバイスを動作させるために，グローバルプログラムから個々のユビキタスデバイス用のローカルプログラムを生成する必要がある．そこで，本研究では，言語処理をし，各ユビキタスデバイス用のプログラムを生成するジェネレータを C ＃を用いて作成し，集中型制御方式と中間型制御方式でプログラムを生成する．各制御方式での個々のユビキタスデバイス用のプログラムへの生成方法について説明する．

- 集中型制御方式
  マスタデバイス用のローカルプログラムとしては，グローバルプログラムにおいて入出力の関数部分が変換されたプログラムが生成される．ここで，マスタデバイスの入出力を制御する場合は，入出力デバイスの ID を取得し，それを制御するためのプログラムに変換される．一方，スレーブデバイスの入出力を制御する場合は，スレーブデバイスの ID とその入出力デバイスの ID を取得し，その制御方法をメッセージとしてスレーブデバイスに送るためのプログラムに変換される．スレーブデバイス用のローカルプログラムとしては，マスタデバイスから送られたメッセージに応じて，対応する ID を有する入出力デバイスの制御を行うプログラムが生成される．3.2 節で説明した関数から集中型制御方式で各ユビキタスデバイス用に生成されるプログラムの主なものを表 2 にまとめた．ある図形上にある LED を光らせる場合は，マスタデバイスは関数 getID と getLocalID により，図形上にある LED を有するユビキタスデバイスの ID とそのデバイス上で光らせる LED の localID をそれぞれ取得し，取得した ID のユビキタスデバイスに光らせる命令を送る．localID はユビキタスデバイス上の 4 つの LED を識別するための ID であり，左上，右上，左下，右下の順に 0，1，2，3 とした．また，スレーブデバイスの入力制御の関数の場合は，マスタデバイスからの命令を受けると，入力データを送り返す．集中型制御方式でのプログラム生成の例を図 8 に示す．この例では，LED4 × 4 の電飾アートの line を点灯させるというプログラムを記述した．記述したプログラムの関数部分 (1，3 行目) を読み取り，マスタデバイスと各スレーブデバイス用のプログラムを生成する．関数部分以外の記述したプログラム (2 行目) は，マスタデバイスのみに書き込まれる．スレーブデバイスの 1, 2, 6 行目はあらかじめ書かれるプログラムである．
- 中間型制御方式
  グローバルプログラムを各ユビキタスデバイスで処理を行うローカルプログラムに分解し，各ユビキタスデバイスに組み込まれる．さらに，1 つのユビキタスデバイスかつ他のすべてのユビキタスデバイスにあるタイミングで動きを合わせるためのメッセージを送るプログラムが加わる．一方で，メッセージを受けるユビキタスデバイスにはあるタイミングでメッセージを受けるまで待つというプログラムが加わる．3.2 で説明した関数の中からいくつかを例として中間型制御方式で各ユビキタスデバイス用に生成したプログラムを表 3 にまとめた．図形上にある LED を光らせる場合は，環境内のすべての

Vol.2012-GN-82 No.28
Vol.2012-CDS-3 No.28
2012/1/20

表 2 集中型制御方式で関数を生成したプログラム

関数	マスタ（上段） スレーブ（下段）

point(*x, y, color*)	ID = getID(x, y);
	localID = getLocalID(x, y);
	if(ID == 0)
	lighting(color, localID);
	else
	Serial.write(data(command, ID, color, localID));

	case command(lighting): lighting(color(data), localID(data)); break;

line(*x1, y1, x2, y2, color*)	getID(line(x1, y1, x2, y2));
	getLocalID(line(x1, y1, x2, y2));
	for(int i=0; i ⟨ num; i++){
	if(ID[i] == 0)
	lighting(color, localID[i]);
	else
	Serial.write(data(command, ID[i], color, localID[i]));
	}

	case command(lighting): lighting(color(data), localID(data)); break;

Read(*x, y*)	ID = getID(x, y);
	if(ID == 0)
	value = analogRead(pin);
	else
	value = request_data(ID);

	case command(analogRead): value = analogRead(pin); Serial.write(data(value)); break;

ユビキタスデバイスが関数 getID と getLocalID で図形上にある LED を有するユビキタスデバイスの ID と LED の localID を取得し，取得した ID が自ユビキタスデバイスの ID と一致したときに点灯させる．中間型制御方式でのプログラム生成の例を図 9 に示す．記述したプログラムの関数部分 (1，3 行目) を読み取り，各ユビキタスデバイス用のプログラムを生成する．関数部分以外の記述したプログラム (2 行目) は，環境内のユビキタスデバイスすべてに書き込まれる．また，1 つのユビキタスデバイスの 1 行目には動き始めの命令を送る処理，他のユビキタスデバイスの 1 行目には命令を待つ処理がそれぞれ書かれる．

表 3 中間型制御方式で関数を生成したプログラム

関数	ユビキタスデバイス (ID = n)

point(*x, y*)	ID=getID(x,y);
	localID=getLocalID(x, y);
	if(ID==n)
	lighting(color, localID);

line(*x1, y1, x2, y2*)	getID(line(x1, y1, x2, y2));
	getLocalID(line(x1, y1, x2, y2));
	for(int i=0; i < num; i++){
	if(ID[i]==n)
	lighting(color, localID[i]);
	}

Read(*x, y*)	ID=getID(x,y);
	if(ID==n){
	value=analogRead(pin);
	send_data(value);
	}
	else
	value = receive_data();

5. 考　　察

電飾アートの用途に応じて，適した制御方式は異なる．そこでグローバルプログラム例をいくつか挙げ，各制御方式での違いを比較し，考察する．

5.1 グローバルプログラム例 1

図 10 に長方形を描く 16 × 16 の電飾アートのグローバルプログラム例を示す．プログラムでは，まずコンフィグレーション部分で LED を 16 × 16 の格子状に配置する．(1) で図形の塗りつぶしを無効にし，(3) で色を設定，(4)-(9) の for 文で長方形を左上から大きくしていき（図 11 上図），(10)-(15) の for 文で長方形を右上から大きくしていく（図 11 下図）．

このプログラムを集中型制御方式，中間型制御方式のそれぞれを用いて実行した．集中型制御方式を用いた場合，点灯させるたびにマスタデバイスがメッセージを送るため，確実な同期制御が行え，運用管理が容易である．しかし，通信量が多くなるため，電力消費が大きい．中間型制御方式を用いた場合，各ユビキタスデバイスがある程度独立して動作するため，通信量が少なく，集中型制御方式よりも電力消費が小さい．わずかにずれが生じることがあるが，一定のタイミングで同期を行うため，グローバルプログラムで記述された通りに

Vol.2012-GN-82 No.28
Vol.2012-CDS-3 No.28
2012/1/20

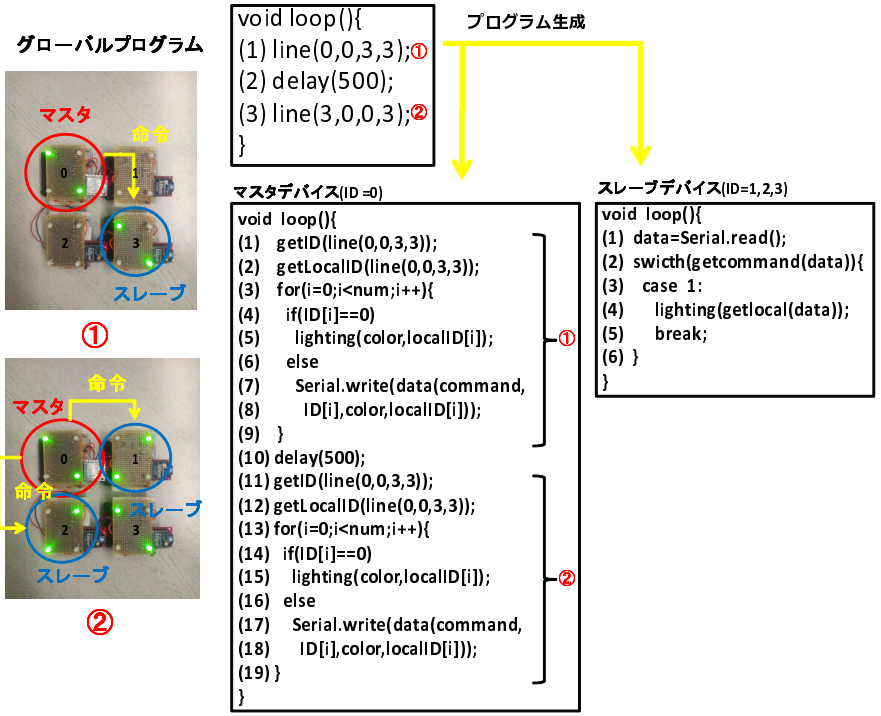

図 8 プログラム生成（集中型）

処理が行われているように見える．以上より，出力のみのグローバルプログラムでは常に同期をとらなくても，タイミングを合わせることは容易であるため，中間型が適している．

5.2 グローバルプログラム例 2

図 12 にセンサを用いた制御のグローバルプログラム例を示す．この例では，すべてのユビキタスデバイスの加速度センサの値を読み取り，閾値を越えているユビキタスデバイスから光を拡散するように点灯させる．プログラムでは，まずコンフィグレーション部分で LED を 14 × 2 の格子状に配置する．(5) でセンサの値を読み取り，(6)-(15) でセンサの値が閾値を越えていればそのユビキタスデバイスから光を両端まで移動させるように点灯させる．

このプログラムを集中型制御方式，中間型制御方式のそれぞれを用いて実行した．集中型制御方式を用いた場合，通信量は多いが，確実な同期制御が行え，運用管理が容易である．

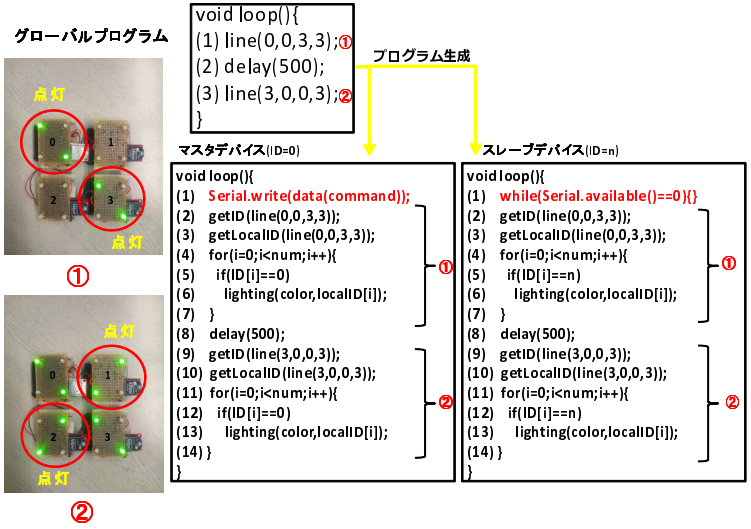

図 9 プログラム生成（中間型）

中間型制御方式を用いた場合，入力処理は出力処理より時間がかかるため，動作のずれが生じやすくなる．しかし，一定のタイミングで修正できるため，グローバルプログラム通りの動作を行える．入力処理が記述されたグローバルプログラムではユビキタスデバイス間の送受信が必要となるため中間型制御方式ではわずかなタイミングのずれで送受信を失敗する可能性がある．そのため，1 つのユビキタスデバイスが他のユビキタスデバイスの入力値を扱う集中型制御方式が適している．

6. おわりに

本研究では，電飾アートに用いるために複数ユビキタス機器の集中型・分散型・中間型制御を用いたマクロプログラミングモデルの設計を行った．設計したマクロプログラミングモデルは，制御対象となるユビキタスデバイスの位置関係，センサや LED などの入出力機器に基づいた制御をマクロな視点で，より柔軟にプログラミングできる． また，実際にいくつかの作品を作成し，集中型制御方式と中間型制御方式で動作させることにより，システム要件に応じて適する制御方式が異なることを確認した．

今後の課題としては，分散型制御方式の実装とグローバルプログラムに応じて最適な制御

Vol.2012-GN-82 No.28
Vol.2012-CDS-3 No.28
2012/1/20

```
void setup()
{
  setposition(0,0,0,1,1);
  setposition(1,2,0,1,1);
  setposition(2,4,0,1,1);
  setposition(3,6,0,1,1);
  setposition(4,0,2,1,1);
        .
        .
        .
  setposition(15,6,6,1,1);
}
```

Arduinoを指定の座標に設置する.

```
void loop(){
(1)  noFill();
(2)  for(int k=0;k<7;k++){
(3)    colorMode(k);
(4)    for(int i=1;i<4;i++){
(5)      rect(0,0,i,i);
(6)      delay(200);
(7)      allLOW(ALL);
(8)      delay(100);
(9)    }
(10)  for(int i=1;i<4;i++){
(11)     rect(5-i,0,i,i);
(12)     delay(200);
(13)     allLOW(ALL);
(14)     delay(100);
(15)   }
(16) }
}
```

色を順番に変える.

左上からLEDが描く四角が大きくなる.

右上からLEDが描く四角が大きくなる.

図 10 グローバルプログラム例 1

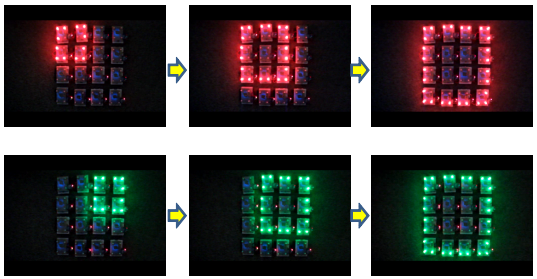

図 11 電飾アート例

方法を自動で選択するといった集中型・分散型・中間型の使い分けなどが挙げられる.

謝　　辞

本研究の一部は，文部科学省科学研究費補助金基盤研究 (A)(20240009, 23240010) によるものである．ここに記して謝意を表す.

参　考　文　献

1) 森川博之: ユビキタスネットワーク社会の構築, 電気学会論文誌C, Vol. 124, No. 1, pp. 12–17, 2004.
2) R. Newton, Arvind, and M. Welsh: Building up to Macroprogramming: An Intermediate Language for Sensor Networks, *Proc. of 4th International Symposium on Information Processing in Sensor Networks (IPSN2005)*, pp. 37–44, 2005.
3) R. Gummadi, O. Gnawali, and R. Govindan: Macro-Programming Wireless Sensor Networks Using Kairos, *Proc. of 1st Distributed Computing in Sensor Systems (DCSS 2005)*, pp. 126–140, 2005.
4) R. Newton, G. Morrisett, and M. Welsh: The Regiment Macroprogramming System, *Proc. of 6th International Conference on Information Processing in Sensor Networks (ISPN2007)*, pp. 489–498, 2007.
5) B. Urs and K. Gerd: A State-Based Programming Model and System for Wireless Sensor Networks, *Proc. of 3rd International Workshop on Sensor Networks and Systems for Pervasive Computing (PerSeNS 2007)*, pp. 261–266, 2007.
6) S. R. Madden, M. J. Franklin, J. M. Hellerstein, W. Hong: TinyDB: An Acquisitional Query Processing System for Sensor Networks, *ACM Transactions on Database Systems*, pp. 122–173, 2005.
7) 木下浩平, 藤田直生, 柳沢 豊, 寺田 努, 塚本昌彦: 分散制御された LED マトリックスを用いた電飾アート制御プラットフォーム, 情報処理学会研究報告, Vol. 2009-EC-12, No. 26, pp. 65–70, 2009.
8) 中田眞深, 児玉賢治, 藤田直生, 竹川佳成, 寺田 努, 塚本昌彦,: プロジェクタによる一斉制御が可能なユビキタス光デバイスの設計と実装, 情報処理学会論文誌, Vol. 50, No. 12, pp. 2871–2880, 2009.
9) 長岡佑典，佐野渉二，寺田　努，塚本昌彦：複数ユビキタス機器を統合的に扱うためのマクロプログラミングモデルの設計，マルチメディア，分散，協調とモバイル (DICOMO2011) シンポジウム，pp. 659-666.
10) Processing, http://www.processing.org/.
11) Arduino, http://www.arduino.cc/.

```
void setup()
{
  setposition(0,0,0,1,1);
  setposition(1,2,0,1,1);
  setposition(2,4,0,1,1);
  setposition(3,6,0,1,1);
  setposition(4,8,0,1,1);
  setposition(5,10,0,1,1);
  setposition(6,12,0,1,1);
}
```

Arduinoを指定の座標に設置する.

```
void loop(){
(1)  int x;
(2)  int k;
(3)  for(int i=0;i<7;i++){
(4)    colorMode(i);
(5)    x=Read(i*2,0);
(6)    if(x>100){
(7)      for(k=0;k<14;k++){
(8)        if(i*2-k>=0 || i*2+1+k<14){
(9)          line(i*2-k,0,i*2-k,1);
(10)         line(i*2+1+k,0,i*2+1+k,1);
(11)       delay(100);
(12)       allLOW(ALL);
(13)     }
(14)   }
(15) }
(16)}
}
```

センサの値を読み取る.

Arduinoから光が拡散していくように光る.

図 12 グローバルプログラム例 2