２００３（平成１５）年度　卒業論文

アクティブロボットビジョンの開発における フレームワークの作成と評価

2004年 2月16日

東京都立科学技術大学

工学部

機械システム工学科

001120　　坂本昌彦

指導教員　臼井清一

目次

第 1 章　緒言

1.1 本研究の背景 1
1.2 本研究の目的 1
1.3 本論文の構成 2
1.4 用語解説 2

第 2 章　システムの構成

2.1 システムの分析とモジュール化 3
2.2 OS とプログラミング言語の選択 5
2.3 フレームワークの提案 6
2.3.1 GSMTD 7
2.3.2 libgsmtdclient ライブラリ 8
2.3.3 libgsmtdarv ライブラリ 9
2.4 フレームワークの使用例 10
2.5 実験・評価方法 11

第 3 章　画像処理

3.1 差分検出 12
3.2 輪郭線と角度検出 14
3.3 近似色重心検出 17

第 4 章　ロボットアーム制御用ドライバモジュールの作成

4.1 使用するロボットアームについて 19
4.2 RS232C（COM ポート）ケーブルの作成 21
4.3 ドライバの作成 23

第 5 章　実験および考察

5.1 計算機システム諸元 25
5.2 移動物体抽出プログラム 26
5.2.1 システム構成 26
5.2.2 実験結果 28
5.2.3 考察 29
5.3 移動物体追跡プログラム 30
5.3.1 システム構成 30
5.3.2 実験 33

第 6 章　まとめ

謝辞

参考文献

第 1 章　緒言

1.1 本研究の背景

アクティブロボットビジョン（ActiveRobotVision:ARV）に用いられている画像処理・画像認識技術はここ十数年で長足の進歩を遂げた．半導体技術の進歩は第一に多様な画像入力機器を生みだし，第二に大幅なコンピュータシステムの計算能力の向上を実現している．

アルゴリズムの面でも遺伝的アルゴリズムやニューラルネットワークの研究は既に峠を越え，システムを構築するに当たって最適なアルゴリズムを取捨選択する組み合わせの時代に入ったと言える．制御対象とユーザーインターフェイスも同様である．

コンピュータシステムも多様化の一途を辿っており，かつては高価なメインフレームでなくては検証できなかったようなアルゴリズムも，安価で高性能な PC 上で実現できるようになった．実際にシステムを構築する際には，FPGA や VHDL に代表されるようなデジタル回路としての実装，あるいは TRON や Linux 等 OS と呼ばれるソフトウェアにまで搭載されるようになっている．

このように画像処理・認識システムは既に実用化のフェーズに入ったが，だからといって果たして画像処理・認識システムの設計から開発に至る一連のプロセスまでもが進歩を遂げたと言えるだろうか．

システムの設計・開発をトータルに眺めるにはプログラミングやアルゴリズムの知識だけでは不十分であり，ハードウェアに関する基礎知識をはじめとして使用する OS の内部構造，現代 OS の提供する機能とそれを利用するためのライブラリの使用法．これら一通りの知識や経験をトータルに眺めなければ，優れたシステム設計と開発プロセスが実現することはない．

最先端技術を研究する大学という場では，これまでそうしたシステム設計・開発プロセスの研究には光が当てられてこなかった．その場限りのコーディングを繰り返したプログラムは，これから研究を志す者にとっては負の遺産となって立ちはだかる．

だからこそ，システム設計・開発プロセスを見直し，柔軟性と拡張性に富んだ再利用性のあるモジュールとフレームワークを作成するべきである．

1.2 本研究の目的

アクティブロボットビジョンシステムを対象とした，柔軟性と拡張性に富み，再利用性のあるモジュールとフレームワークを提案し，実際にシステムを構築し，その効果を検証することが本研究の目的である．

実装はソフトウェア側で行い，検証用のシステムとして移動物体の検出プログラムと，ロボットアームによる動体追跡プログラムを構築する．

1.3 本論文の構成

「第 1 章　緒言」では本研究の背景とその目的について述べる.
「第 2 章　システムの構成」では本研究が提案するフレームワークについて述べる.
「第 3 章　画像処理」では検証用システムで用いる画像処理について述べる.
「第 4 章　ロボットアーム制御用ドライバモジュールの作成」では三菱電機製ロボットアーム「MOVEMASTER EX」を Linux で制御する方法を述べる.
「第 5 章　実験および考察」では検証用システムの構成を説明し，検証結果を示す.
「第 6 章　まとめ」では本研究で実現できたことを確認する.

1.4 用語解説

本研究において頻出する用語と，本研究におけるその意味をまとめておく.

・「モジュール化」
システムの機能を分類し，細分化し，それぞれを独立して動作できるような部品に分解すること.
・「モジュール」
「モジュール化」により独立して機能するようになった部品のこと.
・「コンポーネント」
「モジュール」と同義.
・「フレームワーク」
設計思想とそれを実現するライブラリ群，およびライブラリ群の使用手順．データフォーマットも含まれる.
・「プロトコル」
主に TCP/IP コンピュータネットワークで送受信するデータの書式.
・「Shared Object ファイル」
共有ライブラリと呼ばれ，Windows では DLL にあたる.

第 2 章　システムの構成

2.1 システムの分析とモジュール化

フレームワークを提案するためには，まずシステムのモジュール化を試みる．機能を細分化することで共通する操作を抽出し，機能間でデータを受け渡しするためのフォーマットを考えることがフレームワークへと発展するからである．

本研究ではアクティブロボットビジョン等の画像処理・認識システムでは大別して図 2-1 に示すような三つの機能にまとめることができると考えた．

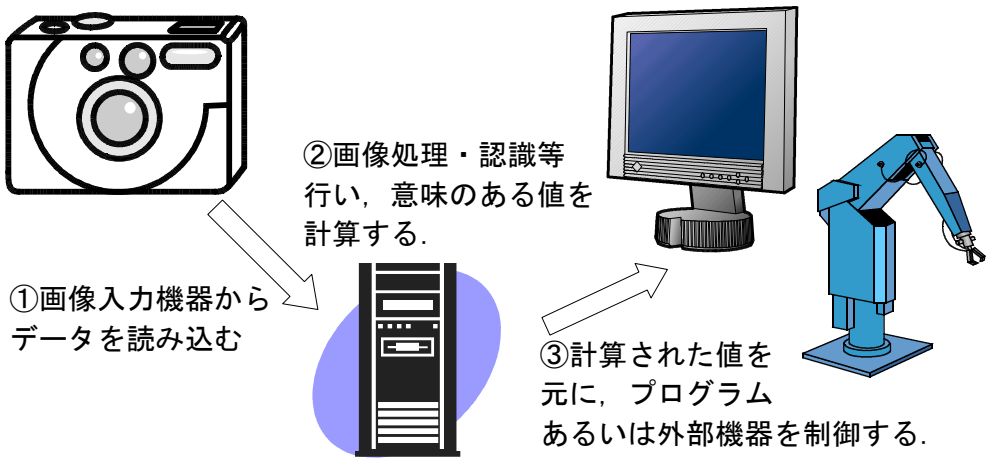


図 2-1 ARV システムに必須の三大機能

1.画像入力機器から画像データを取得する機能
2.画像データを処理し，意味のある値を計算する（例えばボールの位置）機能
3.計算された値を用いて何らかのアクションを実行する機能（ボールの位置をディスプレイ上に表示したり，ロボットアームをその位置へ動かす等）

本研究ではこれら三つを独立したプログラムとして実装することにより，モジュール化を図る．ライブラリとして実装する手法もあるが，モジュールの独立性を極限まで高めるためにこのような方法を採用する．

この手法の利点は各機能が独立したプログラムであるため，不必要な機能は完全に停止させることができるという点である．ディスプレイが無ければ，表示を担当するプログラムを停止させるだけでよい．これによりマシンリソースを環境に応じてなるべく消費しないシステムを構築できる．

別個に分かれたプログラム間でのデータのやりとりには，近年の OS の多くで実装されている「共有メモリ」という機構を用いる．共有メモリとは別々のプログラムからアクセスできるようにした，OS が管理するメモリ領域の事である．

以上をまとめると，本研究で提案するフレームワークの概要は図 2-2 に示すようになった．

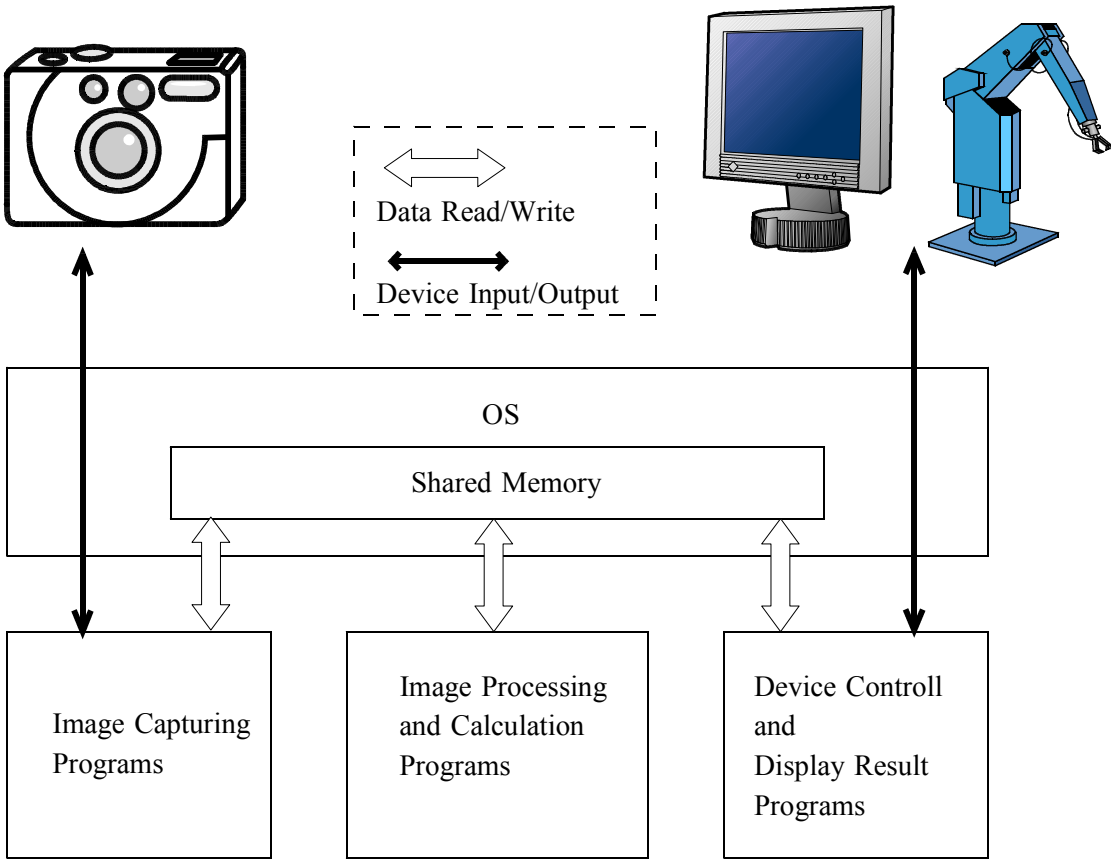


図 2-2 フレームワークの概要

注意点としては，予め画像データのフォーマットやメモリの配置に関して取り決めておくことが重要である．

もう一つ重要な注意点は，画像データの取得機能と処理機能が並列して動作するため，最新の画像データが書き込み終わっていない状態の共有メモリを，処理機能側が読み込んでしまうことがあり得る．これを防ぐには，共有メモリとセットで提供される事の多い「セマフォ」と呼ばれる OS が管理する排他ロック機能を利用して，画像データを書き込んでいる間は他のプログラムがそのメモリにアクセスできないようロックする．

以上をまとめると，本研究が対象とするアクティブロボットビジョンシステムをモジュール化する上で必要なものは次の三つである．

1.複数のプログラムを安定して，安全に実行・管理できるマルチプロセス環境
2.共有メモリとセマフォ
3.これら OS の管理するクリティカルなシステムリソースへ安全にアクセスできるツール・ユーティリティ群

これらを備えたコンピュータシステム，すなわちオペレーティングシステム（OS）を選択する．また，これらの機能を実装するためのプログラミング言語を選択する．

2.2 OS とプログラミング言語の選択

本研究ではオペレーティングシステムとして「GNU/Linux」を採用した．以下にその長所と短所を示す．

【長所】
・共有メモリやセマフォ管理をするためのツールが充実している
・C 言語コンパイラである GCC が無料で利用できる
・デバイスドライバの開発が比較的容易に行える

【短所】
・OS の操作体系や開発環境が Windows と大きく異なる
・そのため，実際に本研究が提案するようなフレームワークを実装できるようになるまでには長い訓練期間が必要である．

この場合の短所は開発者の技能に大きく依存する．本研究では，むしろフレームワークを素早く実装するためのツールの充実と，開発における金銭的なコストの優位性から「GNU/Linux」を採用した．

使用するプログラミング言語には C 言語を用いた．システムリソースにアクセスするのが高級言語中もっとも容易でかつ参考資料も豊富に存在することが主な理由である．

2.3 フレームワークの提案

2.1 で述べたようなモジュール化を実現するにはどのような設計思想と，それを支援するライブラリ群があればよいか考えてみる．それはモジュール化された機能の中から，共通の必須機能を抽出することにより見えてくる．

共通の機能があるということは，ライブラリとしてまとめることができるということである．それにより設計思想が明確になりそれを支援するライブラリが定義できる．

まず必須なのが共有メモリとセマフォの確保である．またそれらに対する一貫した，統一性のあるアクセス方法が必要である．各モジュールが好き勝手な方法でアクセスすると，モジュール間の統一性が低下しソースコードの可読性が著しく損なわれてしまう．

また，ネットワークを用いた分散処理システムへの道を用意する．そのため，ネットワーク機能を用いた拡張性も含める．

上記のコンセプトを元に，本研究では表 2-1 に示す三つをフレームワークを構成するコンポーネントとして提案する．

表 2-1 フレームワークを構成する主要コンポーネント

Component Name	SoftwareType	Main Purpose
GSMTD	Program	Managing Shared Memory and Semaphore
libgsmtdclient	Library	Communicating with GSMTD
libgsmtdarv	Library	Communicating with GSMTD in ARV systems

・GSMTD（Generic Shared Memory Transparent Daemon　汎用共有メモリ透過デーモン）

システムを構成するプログラム群は共有メモリやセマフォにアクセスする．その際，それらシステムリソースに OS が割り当てた識別番号を知っている必要がある．プログラム群は GSMTD に対して TCP/IP 接続による通信を行い，それらの情報を取得する．

・libgsmtdclient（GSMTD クライアントライブラリ）

TCP/IP 接続による定型的な通信手順をまとめたライブラリ．これを用いることにより，TCP/IP 接続に関する知識が無くとも GSMTD と通信し，システムリソースに関する情報を取得できるようになる．

・libgsmtdarv（アクティブロボットビジョン向け GSMTD クライアントライブラリ）

後述するが libgsmtdclient は単に GSMTD との通信を簡略化するだけにすぎない．取得した識別番号を用いて共有メモリやセマフォにアクセスするには煩雑な手続きが必要である．また，ARV システムでは必ず画像データを保存する共有メモリが必要であり，そのためのアクセスも定型的かつ共通である．

そうした，ARV 向けに特化した GSMTD 通信ライブラリが libgsmtdarv ライブラリである．

以下，それぞれのコンポーネントの解説に入る．

2.3.1 GSMTD

Generic Shared Memory Transparent Daemon（汎用共有メモリ透過デーモン）

・共有メモリとセマフォを確保し，TCP/IP ネットワークを通じてそれらの識別番号を通知(GSMTD プロトコル).
・システムのバックグラウンドで動作するサーバープログラム.
・本研究においては画像データと動作パラメータを共有するために使用する.

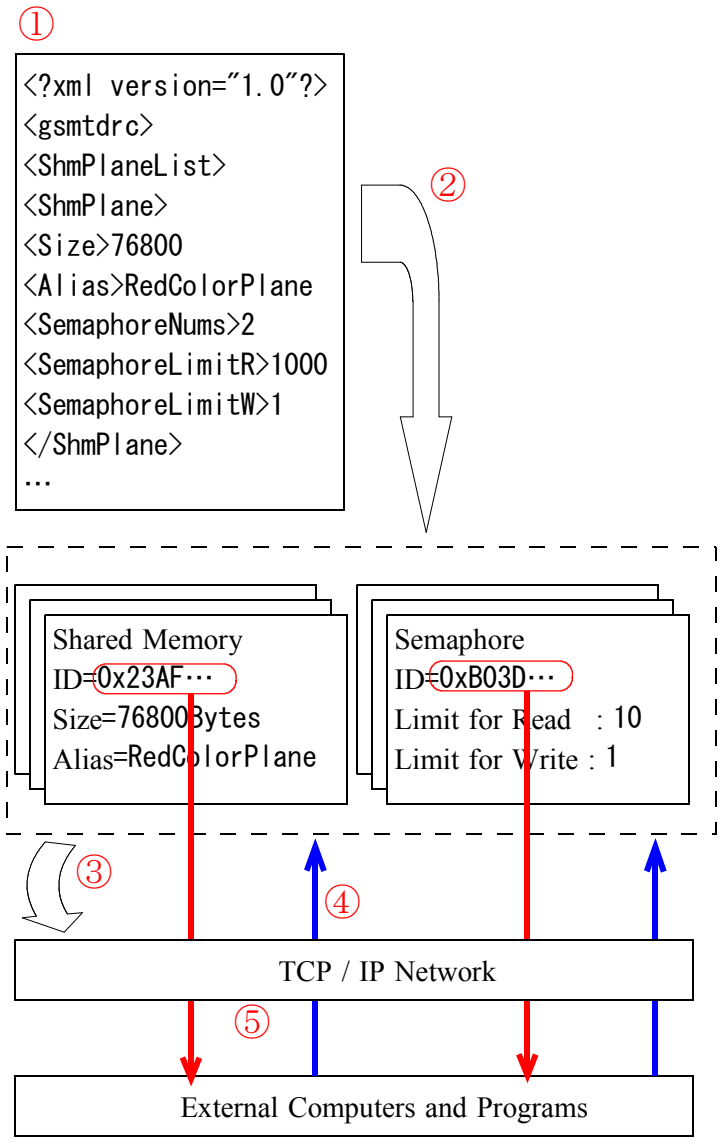


図 2-3 GSMTD の動作

①設定ファイル(XML ファイル)を読み込み，確保すべき共有メモリセグメントの数やサイズ，セマフォ情報を取得する.

②それに従って共有メモリやセマフォ等システムリソースの確保を行う.

③同時に TCP/IP ネットワーク接続の待ち受けも開始する.

④外部プログラムまたはコンピュータシステムが GSMTD の管理する共有メモリやセマフォを利用したい場合，GSMTD プロトコルで共有メモリの識別 ID を通知するよう，GSMTD に要求する.

⑤ GSMTD は要求のあったシステムリソースの識別番号を GSMTD プロトコルで返答する.

（⑥この後，取得した ID を使ってシステムリソースにアクセスする.）

2.3.2 libgsmtdclient ライブラリ

GSMTD クライアントライブラリ（Shared Object ファイル：libgsmtdclient.so）

・GSMTD と通信するための定型的な処理を関数として提供する．
・本研究においては，GSMTD の試験用および後述の libgsmtdarv の基盤として活用する．

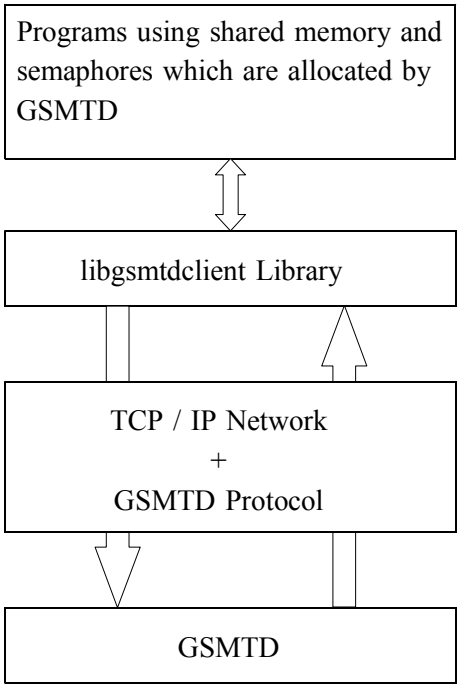


図 2-4 libgsmtdclient の位置づけ

以下に示す定型的な GSMTD との通信機能を関数にまとめた．

・TCP/IP 接続
・GSMTD の設定ファイルに書かれた「別名（Alias）」により共有メモリとセマフォ ID を取得
・共有メモリのサイズを取得
・セマフォの情報を取得

2.3.3 libgsmtdarv ライブラリ

ActiveRobotVision GSMTD クライアントライブラリ(Shared Object ファイル：libgsmtdarv.so)

・libgsmtdclient は GSMTD との通信を簡略化するためのライブラリであるのに対し，libgsmtdarv は GSMTD の管理する共有メモリ・セマフォへのアクセスを簡略化するためのライブラリ.

・本研究においては，GSMTD との通信から共有メモリ・セマフォ操作に至るまでのフレームワークを構成する重要ライブラリ.

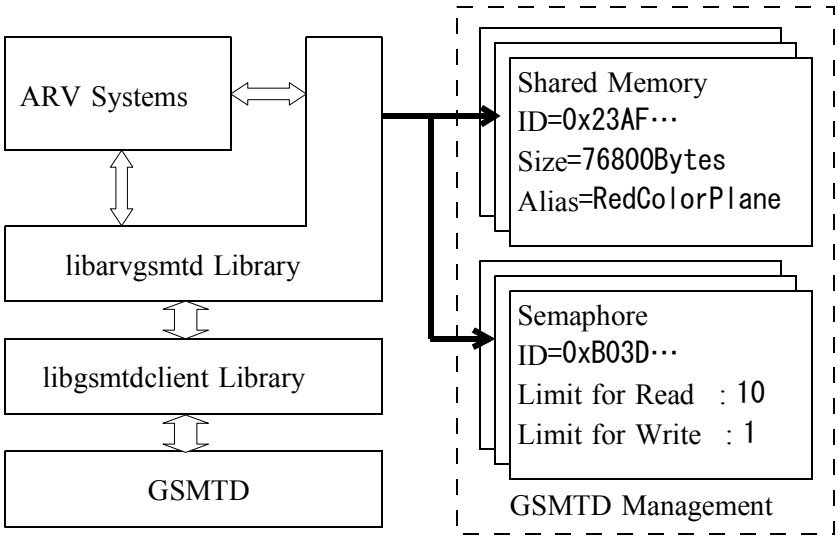


図 2-5 libgsmtdarv の位置づけ

2.4 フレームワークの使用例

以下に,これまで述べてきたフレームワークを用いた大まかなプログラムの流れを示す.

```
#include <stdio.h>
#include <glib.h>

#include "libgsmtdclient.h"
#include "libgsmtdarv.h"

int main(void)
{
        // 共有メモリ・セマフォ情報構造体のリスト(の先頭ポインタ)
        GSList *agsp_head = NULL;
        // 共有メモリ・セマフォ情報構造体のポインタ
        ArvGsmtdShmPlane *agsp_buffer;
        // メモリアクセス用ポインタ
        char *shm_buffer;

        // GSMTDと接続, 共有メモリとセマフォ情報を読み込む
        // 引数は順にポート番号, GSMTDのホスト名, GSMTD設定ファイル,
        // 共有メモリ・セマフォ情報構造体のリストの先頭ポインタ.
        InitArvGsmtdShmPlanes(5000, "localhost", "gsmtdrc.xml", &agsp_head);

        // ArvGsmtdShmPlaneポインタを初期化
        // 共有メモリにつけられた別名(Alias)を使って取得できる.
        agsp_buffer = GetArvGsmtdShmPlaneByAlias("ShmPlaneNo1", &agsp_head);

        // セマフォ(Semaphore)による読み込み操作用排他ロック開始
        ArvGsmtdSemOpRead(agsp_buffer, SEM_LOCK_DOWN);
        // 共有メモリの先頭アドレスを取得
        shm_buffer = (char*)agsp_buffer->Addr;
        ...
        // 読み込み操作用排他ロック終了
        ArvGsmtdSemOpRead(agsp_buffer, SEM_UNLOCK_UP);

        // 書き込み操作用排他ロック開始
        ArvGsmtdSemOpWrite(agsp_buffer, SEM_LOCK_DOWN);
        ...
        // 書き込み操作用排他ロック終了
        ArvGsmtdSemOpWrite(agsp_buffer, SEM_UNLOCK_UP);

        // 共有メモリとセマフォ関連の後始末
        CleanArvGsmtdShmPlanes(&agsp_head);

        exit(0);
}
```

このように，ネットワーク通信やシステムリソースへのアクセスをわずか数行のコードにより処理できるのがモジュール化の強みである．またコードの書き方もほぼ定型的であり，プログラムごとにカスタマイズする必要があるのは読み込む共有メモリと，読み書き処理だけとなる.

これによりシステムを構成するプログラム群に統一感がもたらされ，ソースコードの可読性やバグフィックスを初めとする保守作業が簡単になる.

2.5 実験・評価方法

本研究の目的は ARV システムの構築に適したフレームワークの提案とその検証である．
フレームワークの有用性の検証方法として，本研究では次の二点に着目した．

1.プログラム実行時のマシンリソース（CPU 処理時間，メモリ消費量）の消費量
2.プログラム開発期間

フレームワークは処理を幾重にも覆い隠すもので，それにより CPU やメモリへの負担が大きくなる場合が多い．いくらソースコードの保守性や再利用性が高かろうと，計算時間の増大によりシステムの処理能力が低下すればそれも無意味である．
そのため，本研究ではフレームワークを使用したバージョンと，使用せずに構築した二つのバージョンを用意し，マシンリソースの消費量を比較してみる．
題材としては，差分検出による動体検知プログラムを取り上げることにした．
プログラム開発期間に着目した理由は，近年の組み込み機器に対する開発要求の増大による．すなわち，高度な計算機能力を持たない組み込み開発においては，本研究が提案するフレームワークのようにマルチタスクと共有メモリを潤沢に使用できない環境も多い．
その場合，フレームワークを使用して開発したシステムを，フレームワークを使用しないシステムに作り替える必要が出てくる．どれだけ素早くフレームワーク部分を削り落とし，シングルタスクの「一体成形」型プログラムに落とし込めるかが重要になってくる．
題材としては前述の動体検知プログラムを利用する．
流れをまとめると，最初にフレームワークを使用したバージョンを作成し，続いてそれを一体成形型のフレームワーク未使用バージョンに落とし込む．その際にかかった時間を大ざっぱに求める．そしてできた二つのバージョン間で，マシンリソースの消費量を比較する．
最後に三点目として次を挙げる．

3.Linux を OS として選択した事によるフレームワークの可能性

Linux は非常に柔軟性と拡張性に富む OS として近年勃興してきた．特にデバイスドライバの作成が比較的容易な点を，著者は古いシステムに対する大きな利点として注目している．
古いデバイスでは最新の汎用 OS では対応しきれない仕様も多い．そこで Linux を新旧間の橋渡しとしての可能性と考えた．
本研究では三菱製ロボットアームの MOVEMASTER EX を使用した動体追跡プログラムをフレームワークを用いて作成し，Linux を OS として選択した事による可能性の広がりを実証したい．MOVEMASTER EX は NEC の PC-98xx 時代のシリアル通信による制御方式を用いているため，最新の WindowsOS では通信プロトコルが異なり使用できない．そこで，Linux を用いて MOVEMASTER EX 用の通信ドライバを作成し，作動させる．

第 3 章　画像処理

本章では，本研究で作成する ARV システムで用いる画像処理について説明する．

使用する画像入力機器は「5.1　システムの構成」で述べる通り，CREATIVE 社の USB 接続 Web カメラである WEBCAM PRO eX を用いる．本章で必要な情報は，このカメラから取得できる画像サイズであり，それは横 320 ピクセル，縦 240 ピクセルである．

カメラから入力された画像は RGB 成分に分離され，GSMTD の管理する共有メモリに格納される．

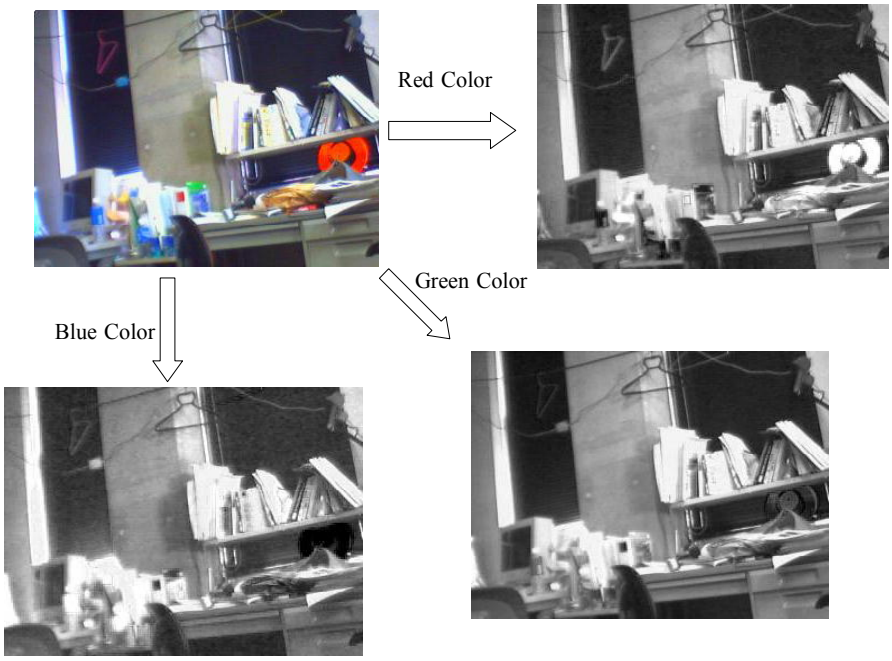


図 3-1 入力画像の RGB 成分への分割

3.1 差分検出

差分検出による動体検知プログラムで用いる差分検出処理について説明する．

差分検出では画像を縦横 10 ピクセルごとに分割し，横 32 個，縦 24 個の計 768 個のセルに分割して処理する（図 3-2）．

各セルに対して，対応する RGB データのピクセル値を合計し，その平均値を計算する．

Pa を平均値とし，*Pi*(*x,y*) をそのセルにおけるピクセル座標 (*x,y*) におけるピクセル値 (0-255) とする．*i* は R，G，B それぞれのデータを表し，1 から 3 までをとる．

w はセルの幅で，*h* をセルの高さとすると，対象となるセルのピクセル平均値は次の式で表現できる．

$$P_a = \frac{\sum_{i=1,2,3} \sum_{x=0,y=0} P_i(x,y)}{3wh} \qquad (1)$$

3wh はセルを構成するピクセルの数で，今回の画像では全セルで $w = 10$，$h = 10$ となり，よって $3wh = 300$ で固定される．

続いて前回の平均値 *Pp* との差分をとり，もし閾値を超えていたら差分が検出されたセルとして，差分検出用の共有メモリエリアに 1 を書き込む．検出されなければ 0 を書き込む．

$$Differ = \begin{cases} 1 : |P_p - P_a| > Threshold \\ 0 : |P_p - P_a| \leq Threshold \end{cases} \qquad (2)$$

差分が検出されたセルを黒く塗りつぶした画像を，図 3-3 に示す．

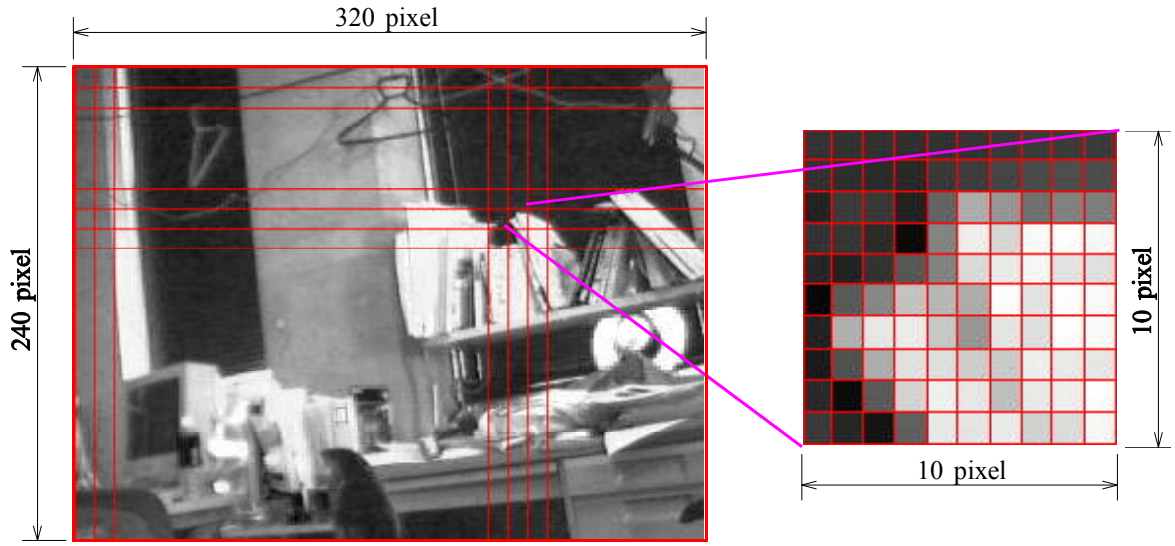


図 3-2 差分検出のためのセル分割

図 3-3 差分検出セルの表示

3.2 輪郭線と角度検出

動体検知プログラムにおいて，フレームワークの拡張性と柔軟性を検証する目的で簡単な輪郭線と輪郭線の角度検出モジュールを作成した．動体検知に直接は使われていないが，現実に ARV システムを構築する際は必須の機能と思われるため，後述する実験時にはこのモジュールも同時に動作させている．

そのモジュールで使用されている輪郭線と角度検出処理について説明する．

輪郭線検出は単純に隣接ピクセルとの差分が閾値を超えたか否かにより処理する．

最終的には一画面に輪郭線を書き込むが，RGB それぞれで二つ以上輪郭線が検出されたピクセルを最終的な輪郭線検出ピクセルとする（多数決）．

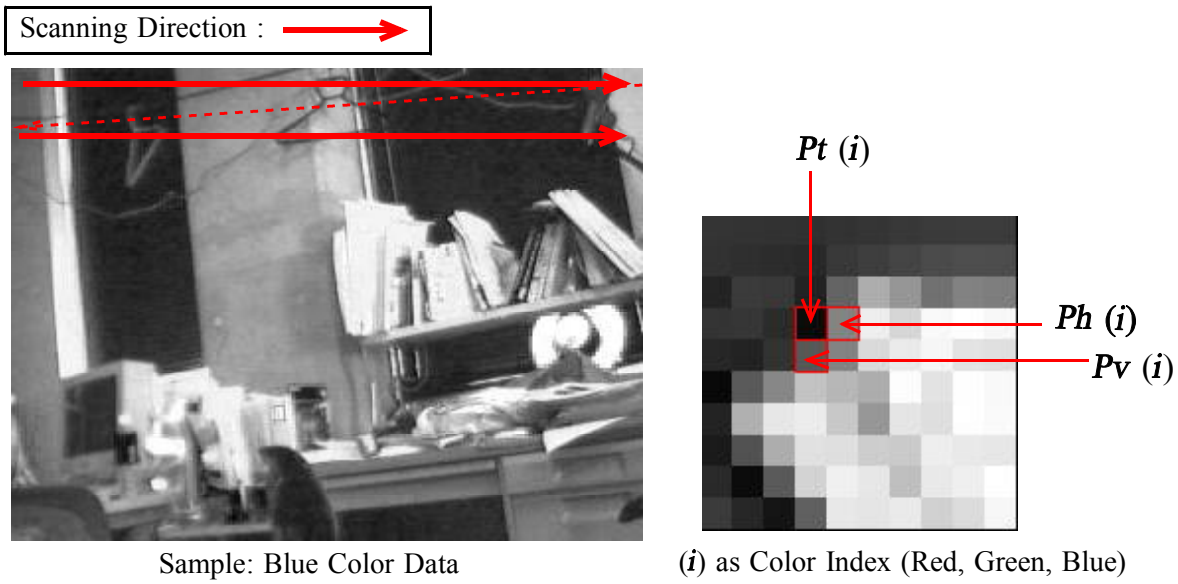


図 3-4 輪郭線検出

対象ピクセル *Pt* が輪郭線に含まれるか否かは，図 3-4 に示すように，そのピクセルの右のピクセル *Ph* と，下のピクセル *Pv* とのピクセル値との差をとり，どちらかが閾値 Threshold を超えれば輪郭線に含まれると判定する．

R，G，B 成分それぞれの画像についてこの判定を行い，２つ以上の画像で輪郭線に含まれると判定されれば，最終的にそのピクセル位置は輪郭線であると判定される．

最終的に輪郭線部分のみを描いた一枚の画像を，GSMTD の管理する共有メモリエリアに格納する．

次に輪郭線の角度検出方法について説明する（図 3-5）．

前述の輪郭線検出画像を，差分検出時と同様に縦横 10 ピクセルで 768 個のセルに分割し，それぞれのセルにおける輪郭線の角度を計算し，大まかに 8 種類に分ける．

角度の計算には，まずそれぞれのセルで，輪郭線検出ピクセルの位置から最小二乗法で直線回帰式を求める．

続いて直線回帰式の係数 *a* から傾きを求め，それを大きく 8 種類に分けて，GSMTD の

管理する角度検出用共有メモリエリアに書き込む.

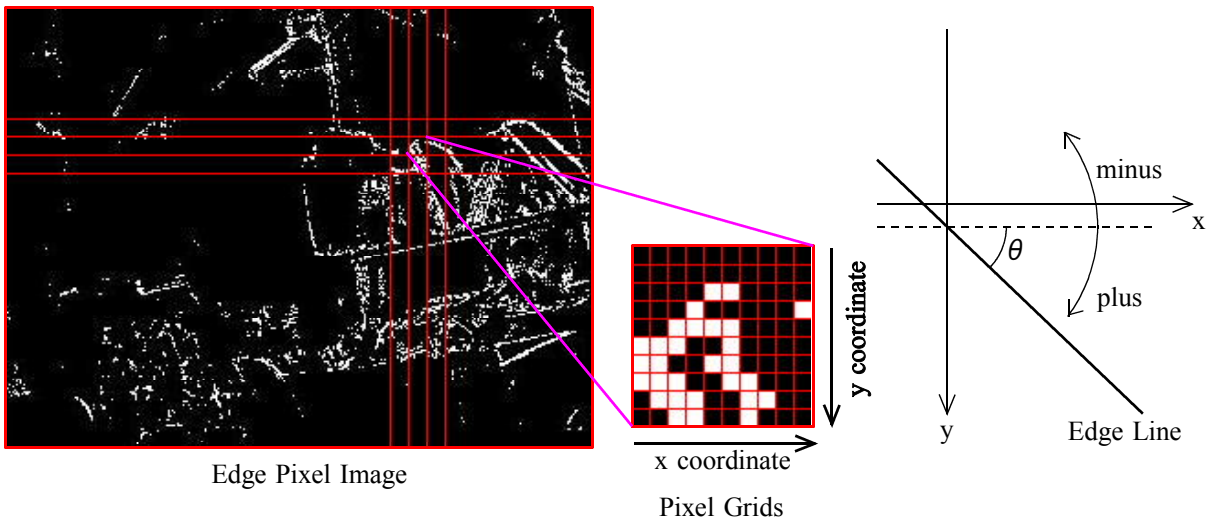


図 3-5 角度検出

$$y = ax + b$$

$$a = \left(\sum xy - \frac{1}{s} \sum x \sum y \right) \Bigg/ \left\{ \sum x^2 - \frac{\left(\sum x \right)^2}{s} \right\} = \tan\theta \qquad (3)$$

（s は対象セル内における輪郭線検出ピクセルの数）

角度の分割は表 3-1 のようにした.

表 3-1

$a = \tan\theta$	by degree（≒）	value
-∞ ... -100.0	-90	7
-100.0 ... -1.19	-89 ... -50	0
-1.19 ... -0.57	-50 ... -30	1
-0.57 ... -0.01	-30 ... -1	2
-0.01 ... 0.01	-1 ... 1	3
0.01 ... 0.57	1 ... 30	4
0.57 ... 1.19	30 ... 50	5
1.19 ... 100.0	50 ... 89	6
100.0 ... ∞	90	7

図 3-6 は輪郭線と角度検出の画像サンプルである.
サンプルでは検出された角度を 8 種類の白黒の濃淡で表している.

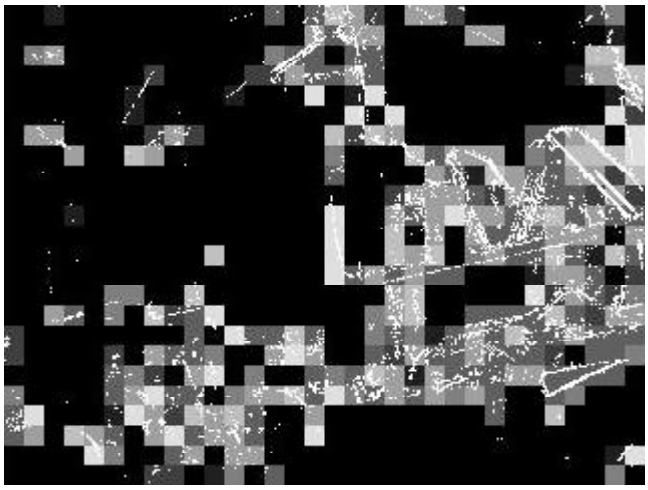


図 3-6 輪郭線検出ピクセルと角度検出濃淡の合成画像

3.3 近似色重心検出

ロボットアームによる動体追跡プログラムにおいては，前述の差分検出による動体検知ではなく，標的判別として色を使用している．標的の色と近似な色の座標重心を標的の位置としている．

ここで用いる近似色重心検出処理について説明する．

まず RGB 成分の画像それぞれについて今までと同様, 縦横 10 ピクセルのセルに仕切り，それぞれのセルごとにピクセル値の平均を算出する．

次に，ユーザーのマウス入力により指定された標的セルの RGB それぞれのピクセル値の平均をとりだし，左上のセルから順次比べていく．

RGB それぞれの差分が，予め指定されている閾値より小さければ，近似色のセルと見なす．

近似色のセルの重心位置 X，Y は以下の式で求められる．

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \frac{1}{N}\begin{pmatrix} \sum x \\ \sum y \end{pmatrix} \qquad (4)$$

N は近似色と見なされたセルの数，x，y は同セルの左上を原点とした位置である．

図 3-7, 3-8 に計算結果のサンプル画像を示す．

図 3-7 がピクセルが平均化された画像である．

図 3-8 は実際の画面で，グレースケールで表示されたセルが近似色として見なされたセルであり，その重心位置のセルが赤い枠で囲って表示されている．（緑色の矩形はロボットアーム作動開始閾値を表している）

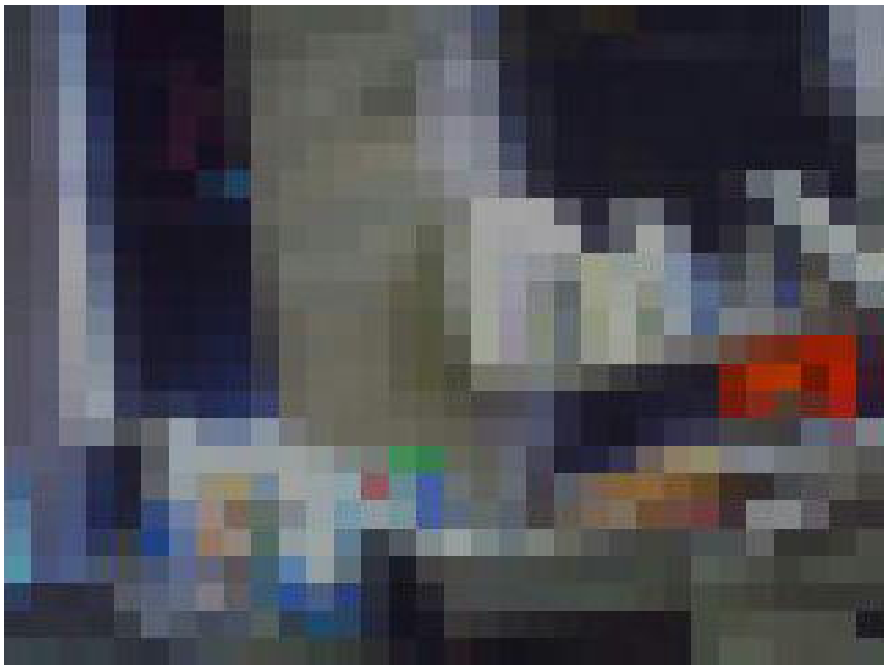


図 3-7　セルごとのピクセル平均値画像

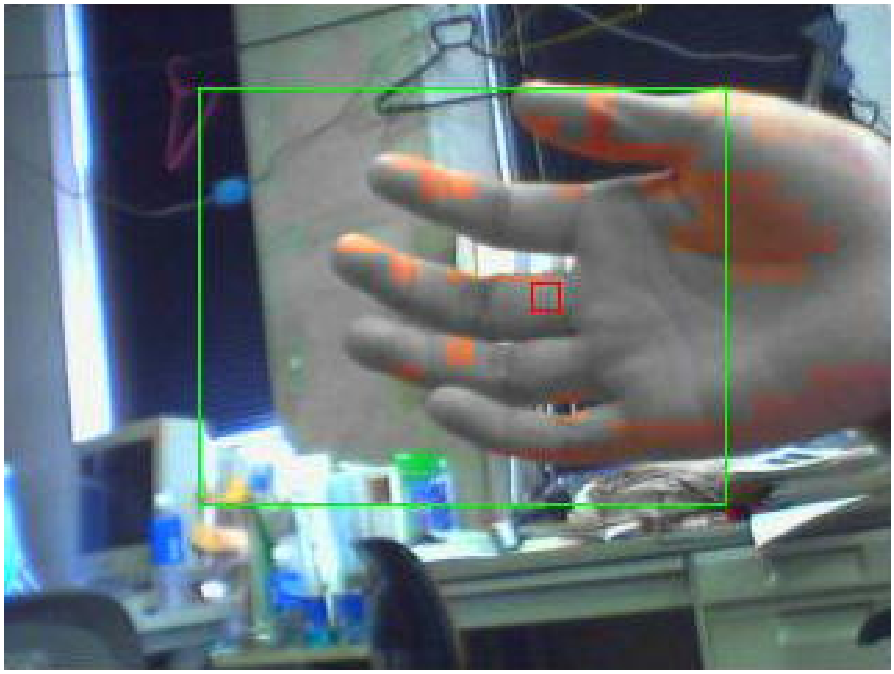


図 3-8　近似色の重心検出画像

第 4 章　ロボットアーム制御用ドライバモジュールの作成

4.1 使用するロボットアームについて

本研究で用いるロボットアームは三菱電機マイクロロボット”ムーブマスター EX”である（以下，MOVEMASTER と呼称）.

MOVEMASTER EX 構成情報
ドライブユニット：D/U-M1
ロボット本体：RV-M1（図 4-1）
電動ハンド：HM-01

図 4-1　MOVEMASTER EX ロボットアームとティーチングボックス（下部コントローラ）

製造時期は 1988 年から 2001 年 3 月までで，2004 年現在，既に生産終了となっている．また修理対応期限も 2008 年 3 月までと三菱電機の産業用ロボットのホームページに掲載されており，旧式と言って良い．
（http://wwwf2.mitsubishielectric.co.jp/melfansweb/robot/details/index_j.htm）

MOVEMASTER は図 4-2 に示すように，外部ドライブユニットに対してパラレル又はシリアル通信を使用して，パソコンからコマンド文字列を送信することにより，アームを制御するようになっている．

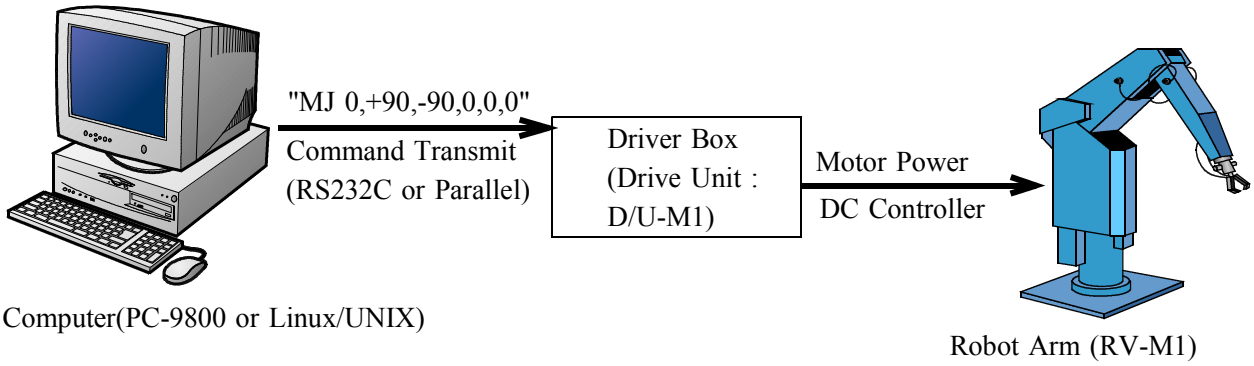


図 4-2　MOVEMASTER ドライブユニットの概要

このロボットアームが製造された当時，日本では日本電気製の PC-98 シリーズと呼ばれる日本独自規格のパーソナルコンピュータが一般に使用されていた．

そのため，ドライブユニットと PC を接続するケーブルのコネクタも PC-98 シリーズに合わせて設計されている．しかし現在用いられている PC/AT 互換機においてはシリアルポートの RS232C，およびパラレルともにコネクタ形状が違うため，そのままではケーブルを接続できない．

そこで本研究では接続ケーブルを自作する必要が生じた．今回は結線数が全部で 9 本の RS232C によるシリアル通信ケーブルの自作を選択した．パラレル通信だと結線数が増加するため自作の難易度が高くなるためである．

もちろん，PC-98 シリーズと PC/AT 互換機とを結ぶためのシリアルケーブルも市販されていた．当初はそちらを使おうと試行錯誤したが，後述するようにシリアル通信時のハンドシェイクで，現在使われていない信号線を MOVEMASTER が使っており，市販シリアルケーブルではそれは結線されていないため，使用できないことが判明した．

次の節以降で，MOVEMASTER のシリアル通信について解説する．

4.2 RS232C（COM ポート）ケーブルの作成

RS232C の詳細については参考文献の「トランジスタ技術 SPECIAL　No.72　パソコン周辺インターフェイスのすべてⅢ」[3]を参照のこと．

本研究で重要だったのは，MOVEMASTER は次の表 4-1,4-2 に示す信号線の内，送受信用の RxD, TxD 線だけでなく，DTR/DSR 線も用いていたという点である．

表 4-1 D-Sub9 ピンコネクタ（PC/AT 互換機）信号線

PinNo.	SigName	Description（日本語）
1	DCD	Detect Carrier Data（キャリア検出）
2	RxD	Receive Data（受信データ）
3	TxD	Transmit Data（送信データ）
4	DTR	Data Terminal Ready（データ端末レディ）
5	GND	Ground（グラウンド）
6	DSR	Data Set Ready（データ・セット・レディ）
7	RTS	Request Transmit Signal（送信要求）
8	CTS	Capable Transmit Signal（送信可能）
9	RI	Ring Indicator（被呼表示）

表 4-2 D-Sub25 ピンコネクタ（PC-9800）信号線

PinNo.	SigName	Description（日本語）
2	TxD	Transmit Data（送信データ）
3	RxD	Receive Data（受信データ）
4	RTS	Request Transmit Signal（送信要求）
5	CTS	Capable Transmit Signal（送信可能）
6	DSR	Data Set Ready（データ・セット・レディ）
7	GND	Ground（グラウンド）
8	DCD	Detect Carrier Data（キャリア検出）
20	DTR	Data Terminal Ready（データ端末レディ）
22	RI	Ring Indicator（被呼表示）

現在市販されているシリアルケーブルは，基本的に送受信とグラウンド線のみを結線しているため，MOVEMASTER との接続には使用できない．さらに，MOVEMASTER はドライブユニットから PC に対してデータを送信する際に RTS/CTS 線も用いていた．

すなわち，PC と MOVEMASTER を接続するにはいわゆる「全線結線」タイプの，しかもクロスケーブルを入手する必要があったが，その探索にかける時間よりも，自作した方が時間がかからないことが予想された．また，その通りになった．

市販品では，クロスケーブルと一口に言っても特に RTS/CTS/DTR/DSR 線周りのクロスの仕方が何種類かあり，実際にどのように結線されているのかはテスタを当てて調べなければならないような状況だったのである．

MOVEMASTER のドライブユニットはパソコンからのコマンド受信だけでなく，ロボットアームの現在の絶対座標値をパソコンに送信する機能もある．

MOVEMASTER のドライブユニットは，PC とのデータ送受信時に，「データ送（受）信の許可」信号をやりとりする．当時は高速なデータの送受信ができなかったため，データが受信可能になったことを送信側にいちいち伝える必要があった．そのための処理は「ハンドシェイク」と呼ばれている．もしハンドシェイク処理を行わず，送信側が勝手にデータを送り続けた場合，受信側では処理しきれず，取りこぼしが発生する．

MOVEMASTER がハンドシェイクに使用する信号線をまとめたものを表 4-3 に示す．

表 4-3 MOVEMASTER ドライブユニットの信号線使用状況

Action	Signal Line Name	
	PC	DriveUnit
PC → DriveUnit	DSR ← DTR CTS ← RTS	
PC ← DriveUnit	DSR ← DTR CTS ← RTS DTR → DSR RTS → CTS	

これに加えて送受信用の RxD，TxD 線と電圧の基準点となるグラウンド線を加え，最終的な RS232C ケーブルの結線状況は下図 4-3 のようになった．

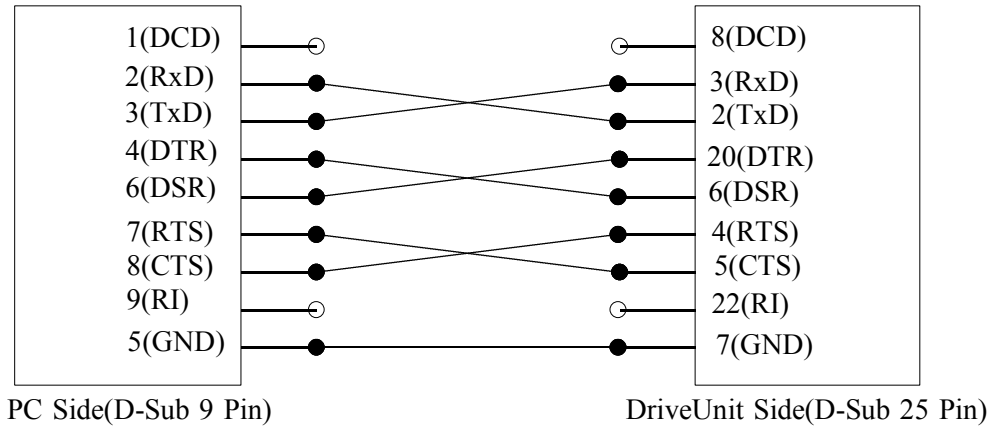


図 4-3　MOVEMASTER 用 RS232C 結線図

4.3 ドライバの作成

前述の通り，MOVEMASTER は現在では使われなくなった信号線を使っているため，ケーブルと同様，OS 側でもシリアルポート用のデバイスドライバを自前で用意しなければならない．

Windows をはじめとする最新の OS では RTS 線などを使用せず，送信・受信・グラウンドの三本のみで通信ができるようなデバイスドライバを採用しているため，MOVEMASTER との通信ができない．

研究当初は Windows で動作させようとしたが，MOVEMASTER 側のデータの送受信部分の記述に困難を極め，結果 Windows を放棄した．

Linux の場合も，Linux カーネルに含まれているシリアルポートドライバでは RTS 線を無視しているため，MOVEMASTER と通信できない．

そのため，Linux カーネルの設定でシリアルポートドライバを無効にした状態でカーネルを再構築したのち，MOVEMASTER 専用のデバイスドライバを自作することにした．

デバイスドライバの作成は表 4-4 に示すタイミングチャートに基づいた．

主な仕様を次に挙げる．

・PC → DriveUnit 時，PC は DSR 線が Hi になる時の割り込み処理で一文字ずつ送信していく．
・DriveUnit → PC 時，PC は RTS を Hi にして DriveUnit に受信可能であることを通知したのち DTR 線が Hi になるまで待機．
・DriveUnit → PC 時，PC は DTR 線が Hi になったら受信を開始するが，PC 側での処理が早すぎて同じデータを取得してしまうため，直前と同じデータであればスキップする．

三番目の仕様に関してはバグの原因になることが容易に予想されうるが，基本的に MOVEMASTER 側から受信できるデータは，ハンドの絶対座標とドライブユニットの RAM 内に保存されているプログラムのみなので，受信動作自体滅多に使うことは無いと考え，問題無しと判断した．

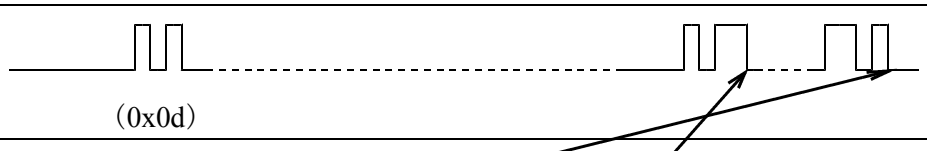
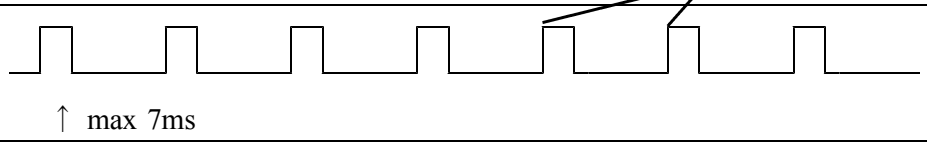
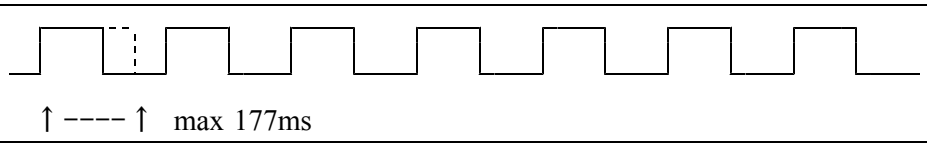
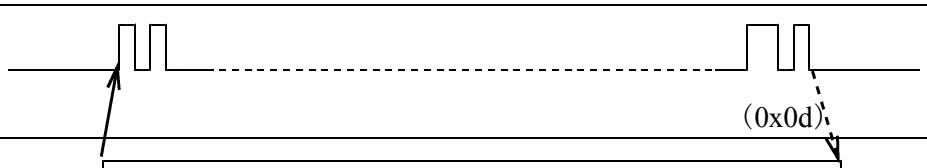
MOVEMASTER　EX は生産ラインで使用することも想定されているため，コマンド送信による角度制御を非常に精度良く実行できる．コマンドでは主に絶対座標か，予め記憶させてある位置番号か，各間接の回転角度を指示するが，いずれも生産ラインで問題なく利用できる程度の精度を実現できる．

従って，MOVEMASTER からのフィードバックは今回の研究では重要ではないと判断した．本研究の目的はあくまでも ARV システムのフレームワークの提案と有用性の検証であり，ARV システムによる MOVEMASTER の位置決め制御ではない．

なお 2004 年 2 月現在，今回作成した MOVEMASTER 専用シリアル通信ドライバモジュールは下記 URL からダウンロードできる．

http://www.c-hino.org/sakamoto/movemaster.tgz

表 4-4 MOVEMASTER EX ドライブユニットの RS232C タイミングチャート

PC	Pulse Signal Timing Chart	DriveUnit
PC → DriveUnit（Command Transmit）		
TxD →		→ RxD
DSR ←		← DTR
CTS ←		← RTS
・DSR-DTR（又は CTS-RTS）線が Hi の間にデータ送信を開始しする． ・Hi になっている間は最大 7ms（DSR）又は 177ms（CTS）．データ送信のタイミングの目安は DSR が Hi の間． ・データ読み込み中，又はコマンド実行中は DTR（CTS）は Low になる． ・コマンド終了コードは 0x0d または 0x0a． ・DSR/CTS が Hi の間に次のデータを送信すると，エラーになる．		
PC ← DriveUnit（Coordinate Data or RAM Program Receive）		
RxD ←		← TxD
DSR ←		← DTR
RTS →		→ CTS
（・PC から，DriveUnit データ読み込みコマンドを送信し，受信準備を始める．） ・PC 側で RTS 線を Hi にし，受信準備完了を DriveUnit に知らせる． ・DriveUnit はデータを送信する前に，DTR 線を Hi にして PC に通知する． ・DriveUnit からのデータ送信が始まる．0x0d が終了コード． ・データ送信が終わると，DTR 線を Low に戻す． ・PC 側も RTS 線を Low に戻す．		

第 5 章　実験および考察

5.1 計算機システム諸元

まず実験に使用した PC の主な仕様を示す.

Panasonic Let's note PRO CF-B5
Intel Pentium 3 700MHz, 192MB SDRAM, 60GB HDD
OS: TurboLinux 8 workstation（kernel-2.4.22）

次に画像入力に用いた USB 接続 Web カメラの仕様を次表 5-1 に示す. また, 外観を図 5-1 に示す.

表 5-1 Creative WEBCAM PRO eX 仕様

カメラハウジング	垂直 60 度/水平 360 度のベース，およびフリップトップレンズハウジング
電源	USB バスパワー
センサ	30 万画素 CCD イメージセンサ
解像度	320x240 他
感度	6 ルクス
ビデオフォーマット	24 ビット RGB
露光コントロール	自動
カラーバランス	自動
カラーマトリックス	プリセットでカメラに積算
ポート互換性	USB 1.1

図 5-1 CREATIVE WEBCAM PRO eX

5.2 移動物体抽出プログラム

最初にフレームワークの計算機のパフォーマンスに対する影響を調べるための，移動物体抽出プログラムについて実験する．

5.2.1 システム構成

下図 5-2 に移動物体抽出プログラムのシステム構成を示す．

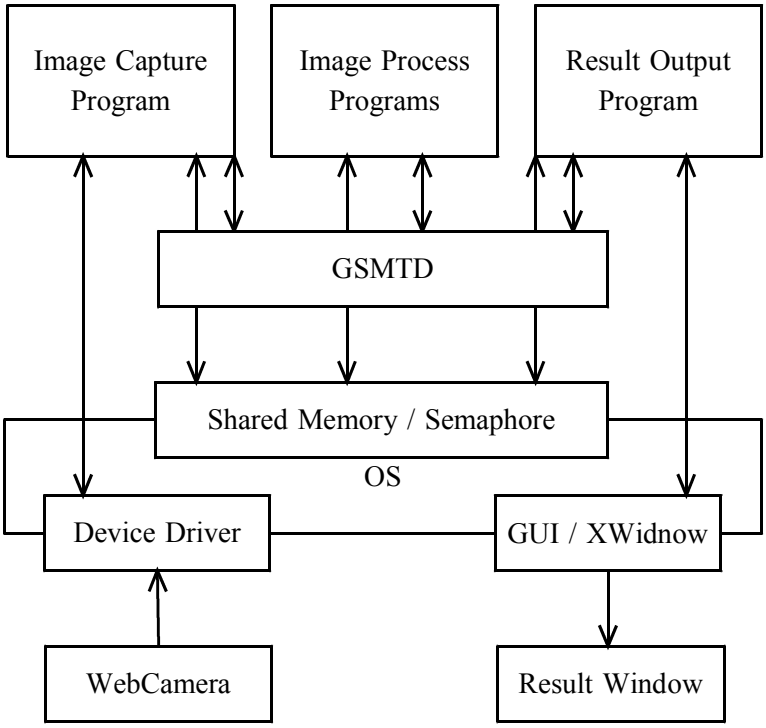


図 5-2 移動物体抽出プログラムのシステム構成

図 5-3 に GSMTD とそれを利用するプログラムの関係を示す．

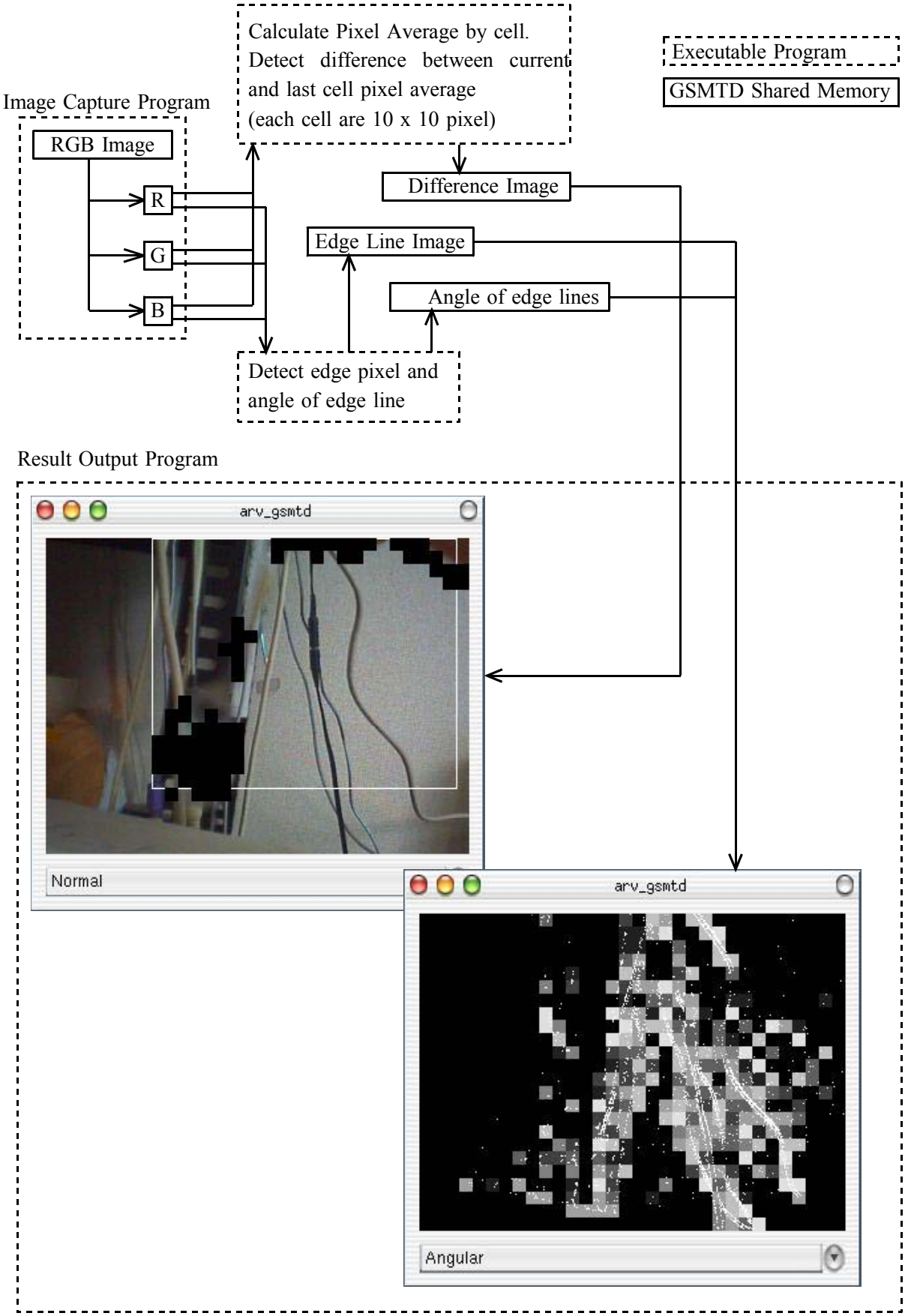


図 5-3 差分検出・輪郭線角度検出処理フロー

5.2.2 実験結果

GSMTD が計算機の全体的な処理性能に与える影響を見るために，今回は OS に付属する XOSView というプログラムを用いて簡易測定した．

そもそも，フレームワークを使用したシステムと使用していないシステムとでは，マルチタスクとシングルタスクの違いが発生するため，CPU の消費量やメモリ消費量など，細かな数値を調べてもあまり意味がない．

むしろ全体の処理性能を大まかに調べる方が理にかなっていると判断し，XOSView というグラフィカルなパフォーマンスメータプログラムを使用することにした．

図 5-4 にシステムが起動する前の XOSView の画面を示す．
図 5-5 にフレームワークを使用したシステムが起動した後の画面を示す．
図 5-6 にフレームワークを使用せずに構築したシステムの場合の画面を示す．

図 5-4 システム起動前の状態

図 5-5 フレームワーク使用版

図 5-6 フレームワーク未使用版

5.2.3 考察

フレームワークの使用・未使用の間で大きな変化は見られなかった．このことは，本研究が提案するフレームワークを使用してシステムを構築しても，計算機のパフォーマンスに大きな影響を与えることが無いことを実証している．

さらに,システムをプログラムレベルで分割することによる副次的な効果もあらわれた．

リアルタイムな処理では，通常はタイマー処理を用いる．本研究の提案するフレームワークでは，モジュールをプログラムレベルで分割することにより，個々のプログラムごとのタイマー設定が可能になった．これにより，画像入力部分ではできる限りタイマー間隔を短くとり，画像処理・結果出力側では多少遅めにすることにより，システム全体として無駄を省くことが可能となる．

フレームワークを用いないシングルタスクでは，スレッド機能を用いない限りは，画像入力から処理・出力までが一連の流れに組み込まれてしまうため，全体のタイマー設定に全てが引きずられてしまう．

続いて，フレームワークを使用したシステムをシングルタスクの未使用版に落とし込む作業に要した時間について述べる．

一日 8 時間の作業として，二日を要した．

これが多いか少ないかは比較対象が無い上，作業者のプログラミング技能に依るところが大きいため何とも言えない．しかし，作業内容としてはフレームワーク処理を削り，一つのファイルにまとめるだけで済んだので非常に簡単であったことは確かである．

逆に最初からシングルタスクで構築したシステムを，フレームワークを抽出して分割する手間を考えれば，明らかに最初からフレームワークを用いて構築し，必要に応じてシングルタスクにまとめる手法を採った方が手間は少ない．

以上より，本研究の提案するフレームワークの有用性は実証されたと考えられる．

5.3 移動物体追跡プログラム

続いて MOVEMASTER を用いた移動物体追跡プログラムについて実験する.

5.3.1 システム構成

図 5-7 に移動物体追跡プログラムのシステム構成を示す.
図 5-8 にカメラを装着した MOVEMASTER の模式図を示す.

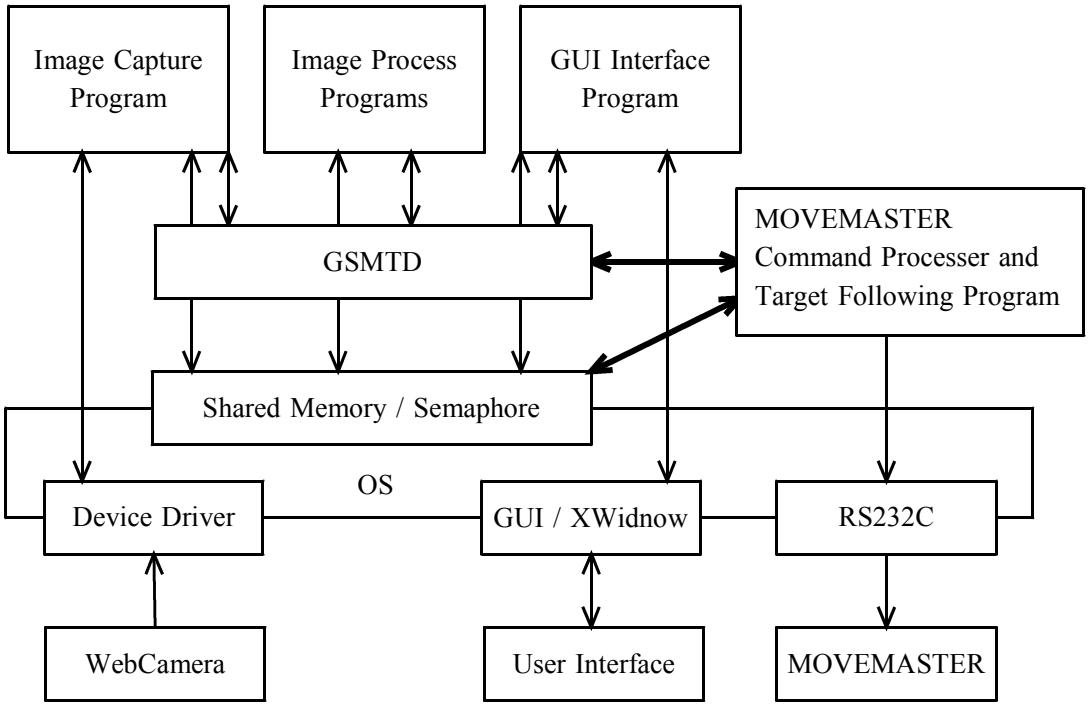


図 5-7 移動物体追跡プログラムのシステム構成

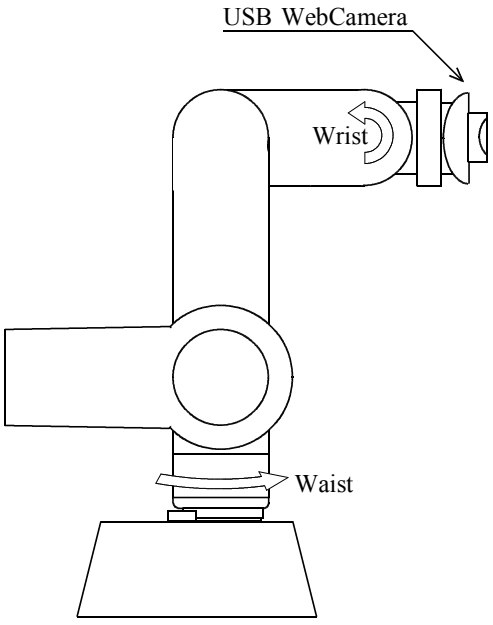


図 5-8 カメラを装着した MOVEMASTER

今回は追跡対象決定のための近似色重心検出処理が加わり，そのパラメータをユーザーが画面上でグラフィカルに決定できるよう，ユーザーインターフェイスと結果表示用のGUI プログラムを作成した．

さらに，重心位置から MOVEMASTER に送信すべきコマンド文字列を生成し，送信するためのプログラムも追加された．

動かす関節は図 5-8 に示すように Wrist と Waist のみに制限した．

図 5-9 に処理フローとユーザーインターフェイスについて示す．

図 5-9 中の画像は，プラスドライバーのオレンジ色のグリップを標的として選択している様子を表している．近似色と判定されたセルがグレースケールで表示され，その重心位置が赤い枠で囲まれている．

緑色の枠線を重心位置が超えると，その方向に MOVEMASTER を動かす．水平方向に超えると Waist 軸が回転し，垂直方向に超えると Wrist 軸が回転する．

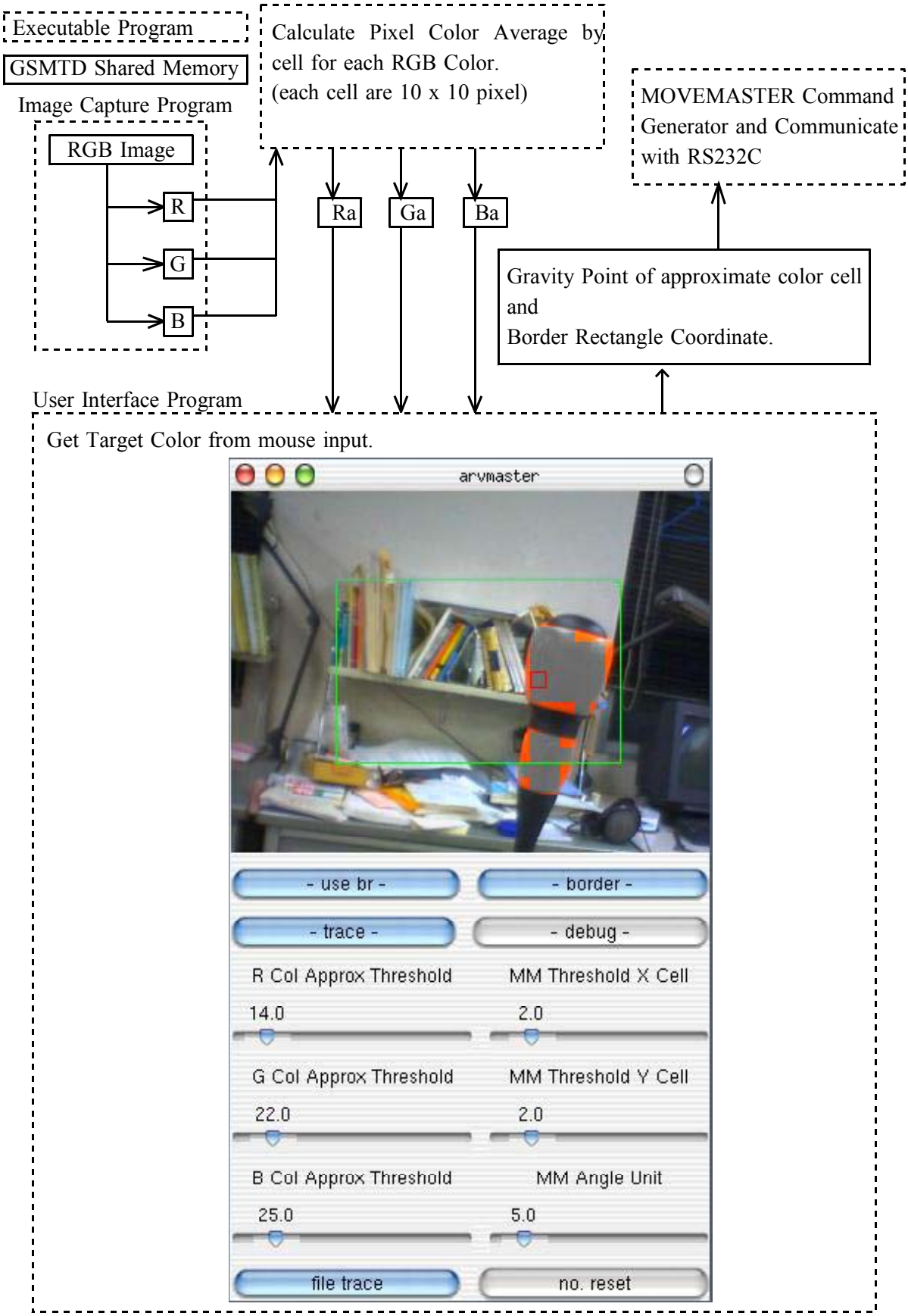


図 5-9 移動物体追跡プログラム処理フロー

5.3.2 実験

図 5-10 に，アーム先端に Web カメラを装着された，追跡プログラム動作中の MOVEMASTER の様子を示す．

図 5-11 に，実際に Waist(水平)方向，Wrist(垂直)方向に追跡させたときの連続画像を示す．背景画像との位置関係に注意してみると，確かにカメラが標的を追って移動していることがわかる．

図 5-10 追跡中のロボットアーム

最終的に，MOVEMASTER を用いた簡単な移動物体追跡プログラムを作成することができた．

近似色を用いているため，背景色と同系統の色を追跡させるのは難しい．また，光線のあたる具合によっても近似色判定が大きくずれてしまうため，パラメータの調整を行う必要があった．

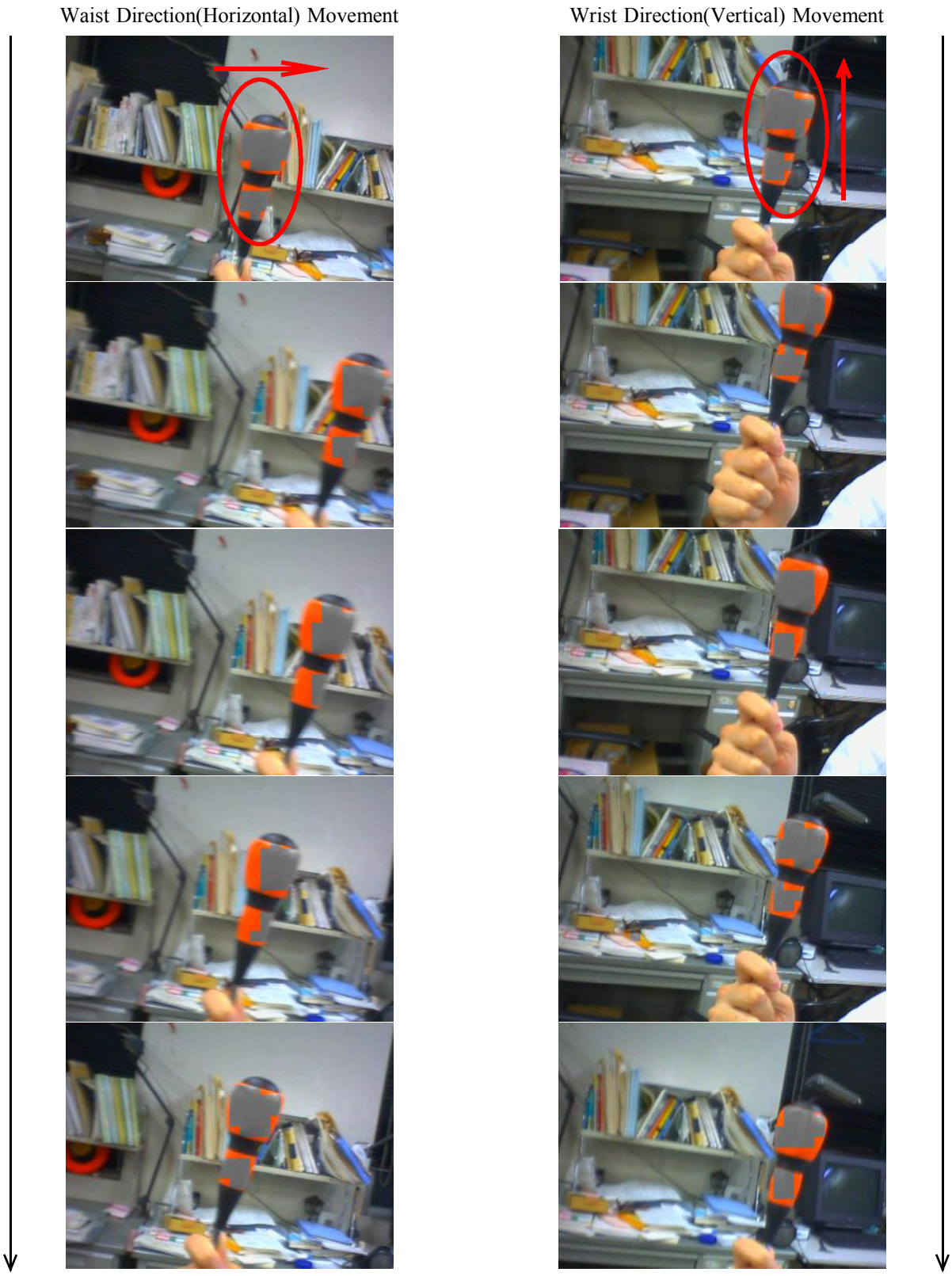


図 5-11 追跡中の連続画像

第 6 章　まとめ

本研究ではアクティブロボットビジョン向けのフレームワークの作成とその有効性の検証を行った.

・フレームワークとして汎用共有メモリ透過デーモンとその周辺ライブラリを作成した.

・OS として Linux を採用したことにより，柔軟なプロセス制御や MOVEMASTER のような旧式デバイスとの通信ドライバの容易な作成が可能になった.

・差分検出による動体検出システムを構築してみた結果，フレームワークを使用しても計算機の全体パフォーマンスに影響を与えないことが確認された.

・最初からフレームワークを使用してシステムを構築することにより，システムを単純化することが容易になることが確認された.

以上により本研究が目的とする，本研究の提案したアクティブロボットビジョン向けのフレームワークの有用性が実証された.

謝辞

敬称略，順不同（研究室，所属等は 2003 年度のものです）

臼井清一先生（機械システム工学科）
村越英樹先生（電子システム工学科）
井上勝彦先生（同上）
臼井研究室の全メンバーとその他の友人
家族

私に計算機システムとプログラミング技法・思想について絶大な影響を与えた心の師匠として，特に次の名前を記す．
坂本弘毅氏，藤原博文氏，西田亙氏

参考文献

(1) 奥脇学，Linux プログラミングガイドブック，(2001)，266，秀和システム
(2) 河野清尊，C 言語による UNIX システムプログラミング入門，(2003)，464，オーム社
(3) 山形孝雄，トランジスタ技術 SPECIAL No.72 特集パソコン周辺インターフェイスの全てⅢ，(2000)，173，CQ 出版社

以下のホームページおよびその作成者に対して感謝する．これらのリソースが無ければ本研究は有り得なかった．(URL は 2004 年 2 月時点のものです.)

・Hardware with Linux / Linux でハードウェア
http://www.mech.tohoku-gakuin.ac.jp/rde/contents/linux/indexframe.html
・Welcome to skyfree.org !
http://www.skyfree.org/
・Turbolinux World : Turbolinux Library
http://www.turbolinux.co.jp/world/library/features/c_magazine/vol_05.html
・RPM-BUILD-HOWTO
http://www.linux.or.jp/JF/JFdocs/archive/RPM-BUILD-HOWTO.html
・Program Library HOWTO
http://www.linux.or.jp/JF/JFdocs/Program-Library-HOWTO/index.html
・C++:language&libraries（cppll）過去ログ
http://www.tietew.jp/cppll/
・The Linux Serial HOWTO
http://www.linux.or.jp/JF/JFdocs/Serial-HOWTO.html#toc3
・The Linux Serial Programming HOWTO
http://www.linux.or.jp/JF/JFdocs/Serial-Programming-HOWTO.html#toc1
・The Linux Modem-HOWTO
http://www.linux.or.jp/JF/JFdocs/Modem-HOWTO.html#toc4
・Welcome to Webcam（USB カメラを使おう!!）
http://keiren-web.hp.infoseek.co.jp/webcam.html
・Linux にビデオカメラ(ウェブカメラ)をつなぐ
http://www.rmatsumoto.org/camera/linux-camera.html
・Linux OVCam Drivers
http://alpha.dyndns.org/ov511/
・Linux drivers for Philips USB webcams
http://www.smcc.demon.nl/webcam/
・GTK v1.2 チュートリアル(日本語版)
http://www.kitanet.ne.jp/~asler/linux/gtk/ja/gtk_tut_ja.html
・P'z Cafe
http://homepage1.nifty.com/kamikaze_attack/
・sci10 - cgtkcalc and programming
http://www.mars.sannet.ne.jp/sci10/
・Larse
http://shimm.hp.infoseek.co.jp/